卒業研究報告書

題目

Java を用いた 5 五将棋 AI の開発

指導教員

石水 隆　講師

報告者

17–1–037–0175

池田 伸晃

近畿大学理工学部情報学科

令和 04 年 01 月 24 日提出

概要

５五将棋は,5 × 5 の盤面を用い,6 種類の駒のみを用いるミニ将棋の一種である．５五将棋は, 盤面が狭く決着が速い, 定石などもなく創造性が試される, 適度に奥が深いの 3 つの特徴がある．５五将棋で使用する駒は, 王将・飛車・角行・金将・銀将・歩兵の 6 種類であり, 本将棋と同じく歩, 銀, 角, 飛は敵陣に入るとそれぞれと金, 成銀, 馬, 龍に成る事ができる. 将棋と同様に, プレイヤーは交互に盤上の自分の駒を 1 つ移動させるか, 相手から取った駒を盤上の空いているマスに置くことができ, 相手玉に王手をかけ, 相手がどの手を選んでも次の手で玉将を取られる状態を回避できない場合「詰み」となる. 既存の５五将棋のアプリケーションはいくつかあり, それらのアプリケーションでは何手か先局面を先読みしその局面での評価値を用いて着手選択するが, 先読み手数を増やすと計算時間が指数的に増えるため先読みできる手数には限界がある.

そこで本研究では, ５五将棋のアプリケーションを作成し, 先読みした局面で敵玉に王手が掛けられる場合, その局面が王手の連続で詰ませることができるかかを判定するルーチンを組み込み, 短手数の詰みを逃さないようにすることで探索の高速化を図る.

目次

1 序論

1.1 ５五将棋とは

５五将棋とは, 亜種の将棋ゲームの一種で,1970 年頃に楠本茂信氏が作ったゲームと言われている [5]．プレー人口はそれほど多くはないが, 大会が行われていたという記録もあることから．それ相応の難易度のゲームであることが知られている．５五将棋は, 本将棋を遊んだことがある人はすぐに楽しめるゲームである．本将棋は, 定跡や戦略を覚えるなど初心者が越えなければならない壁があるが, ５五将棋については, 駒の動かし方を覚えてしまえば誰でもすぐに楽しめるという手軽さがある．

1.2 ５五将棋の既知の結果

５五将棋 AI の大会は, 毎年行われている.[5][10] ここ数年の UEC 杯は, 大会のレベルは向上している一方で上位プログラムの実力は拮抗している傾向にあり, 突出して強いプログラムはまだ登場していない

1.3 将棋 AI の手法

コンピュータ将棋で指し手を決定するときにその局面で着手可能な全ての手に対して，その手から一定手数指した後の局面を生成し, その局面での評価値を指し手選択の目安として用いる方法がある．局面の評価値は, 双方が持つ駒の価値, 駒同士の連携度, 駒の稼働範囲, 玉の安全度などの要素を用いて決定される．この手法を用いる場合, 評価値の計算に, どのような要素を用いるか, 各要素にどれくらいの重みを付加するかが重要なポイントとなる．本将棋ではプロ棋士により, 駒の評価値はほぼ決まっている．一方５五将棋では使用する駒の枚数や盤面の広さが異なるために駒の評価値として本将棋と同じ評価値は適当ではない.[4][6]

局面の先読みには MiniMax 法やその改良版の $\alpha\beta$ 法が用いられる．$\alpha\beta$ 法は、MinMax と同じ結果を得られるにもかかわらず, 理論上の計算量はもっとも条件の良い場合で MinMax と比較した場合, 同じ時間でほぼ二倍の深さまで読める手法である．一般に先読み手数が深くなるほど強くなる [6] が, 先読み手数が増えると探索時間が指数的に増えるため, 適切な枝刈りを行わなければならない．どこまで読むのか, どのような条件で枝刈りするのかがこの手法を用いる場合のポイントとなる．

今回作成した AI では, 柿木による５五将棋の自動学習得られた本将棋の評価値より飛車・と金が少し高い評価値で $\alpha\beta$ 法を行う.[4] また, 先読み手数を増やし, 探索時間を抑えるため敵玉に王手を掛けられる局面では, 王手の連続で敵玉を詰ませることができるかを判定する「詰将棋ルーチン」を用い短手数の詰みを見逃さないようにする．

1.4 詰将棋を解くプログラム

詰将棋とは, 駒が配置された将棋の局面から王手の連続で相手の玉将を詰めるパズルである．詰将棋を解くための手法は様々なものが提案されている．[11, 12, 13, 14, 15, 16, 17].

詰将棋を解くプログラムには AND/OR 木探索,df-pn アルゴリズムなどがある.[19] AND/OR 木探索では, 先手番の局面をを OR 節点, 後手番の局面を AND 節点とし, 各接点は先手勝ち (1)・不詰め (0)・不明 (x) の値を持っている．子節点をもつ節点を内部節点という．子節点を持たず先手勝ち (1)・不詰め (0) の値をもつ節点を終端節点という．終端節点は, 先読みした局面で敵玉が詰んでいる局面 (1) または敵玉に王手を掛けるこ

とができない不詰の局面 (0) である．OR 接点は, 子接点の値が一つでも 1 であれば 1,AND 接点は子接点の値が一つでも 0 であれば 0 となる．このとき, 根となる現在の局面の値が 1 であれば, その局面は王手の連続で詰むことになる．

df-pn アルゴリズムでは, 探索開始局面の証明 ((pn,dn)=(0, ∞)) または反証 ((pn,dn)=(∞,0)) が示されるまでの間, OrNode では pn が最小の局面を,AndNode では dn が最小の局面を選んで探索を進めるアルゴリズムである．

今現在, 詰将棋を解くプログラムの進歩はめざましく, 多数存在するが, 詰将棋を解く探索方法はほぼ確立されている．

1.5 本研究の目的

本研究の目的は, 5 五将棋のアプリケーションの作成し, 将棋 AI の探索の高速化を図ることである．そこで, 詰将棋ルーチンが探索の高速化に有用かを検証する．

1.6 本報告書の構成

本報告書の構成は以下の通りである．2 章では 5 五将棋について解説し,3・4 章では作成したプログラムについて, 5 章では検証結果を示す．

2 ５五将棋

2.1 ５五将棋の概要

５五将棋は,5 × 5 の盤面を用い,6 種類の駒のみを用いるミニ将棋の一種である．５五将棋で使用する駒は, 王将・金将・銀将・飛車・角行・歩兵の 6 種類であり, 本将棋と同じく歩, 銀, 角, 飛は敵陣に入るとそれぞれと金, 成銀, 馬, 龍に成る事ができる．図 1 に 5 五将棋の初期配置図を示す．

図 1　5 五将棋の初期盤面

2.2　5 五将棋のルール

5 五将棋の駒は, 王将・金将・銀将・飛車・角行・歩兵の 6 駒からなり, 駒の動きは本将棋と同じである. 本将棋と同じで, 先手後手は交互に一手ずつ手番が周り, 各手番では盤上にある自駒を動かすか持ち駒を打つことができる. 相手の駒の居る場所に自分の駒を動かした場合, 相手の駒を取って持ち駒とすることが出来る. 持ち駒は自分の手番の時に盤上の駒の置かれていないところなら好きな場所に打つことができる. 双方の陣地は指し手の近傍 1 段で, 歩，銀，角，飛は相手の陣地に入ることによって成ることができる. 次の手で相手の王将を取ることが出来る状態になる手を「王手」と呼び, 相手がどの手を選んでも次の手で玉将を取られる状態を回避できない場合「詰み」となる. 先に相手の王将を「詰み」の状態にした方が勝ちとなる. 禁じ手は, 歩を打つことによって相手の王将を詰ませる「打ち歩詰め」, 味方の歩の居る同じ筋に歩を打つ「二歩」, 盤上の駒を行きどころの無い状態にする「行きどころのない駒」, 王手を掛けられた方が王手を回避しない「王手放置の禁」がある. 本将棋の「千日手」は,, 同一局面が 4 回現れたとき千日手が成立し無勝負となるが, 5 五将棋の「千日手」は, 千日手が成立すると先手側の負けとなる.

3　5 五将棋のプログラム

本章では, 本研究で作成した 5 五将棋 AI について述べる. 本研究では,Java を用いて 5 五将棋 AI を作成した, 付録に本研究で作成した 5 五将棋 AI のプログラムを示す. 本研究で作成した 5 五将棋 AI は,[9] の本将棋の将棋 AI を参考に, 5 五将棋用に改良を行ったものである. 以下に本研究で作成した 5 五将棋 AI のプログラムのクラス図と各クラスについて説明する.

3.1 プログラムの仕様

本節では，本研究で作成した 5 五将棋 AI のプログラムの仕様について述べる．
プログラムを起動させると, 図 8 の画面が表示される．

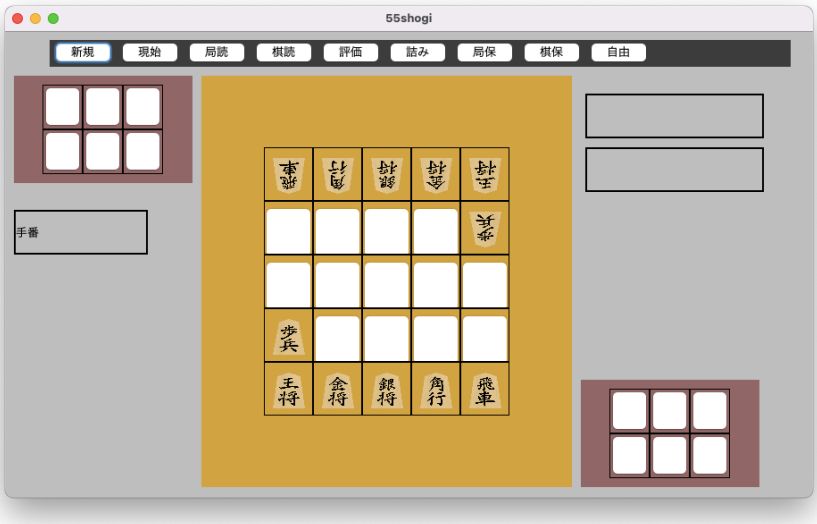

図 2　アプリケーションの初期画面

「新規」ボタンを押すと, 図 3,4,5 の順番に選択画面が表示され, 全ての選択を終えると新規対局を始めることができる．

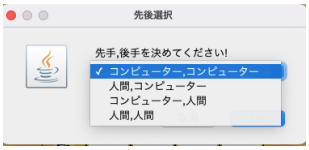

図 3　先後選択

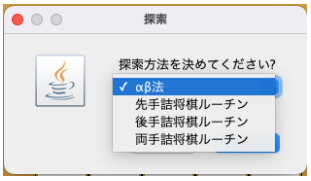

図 4　探索方法選択

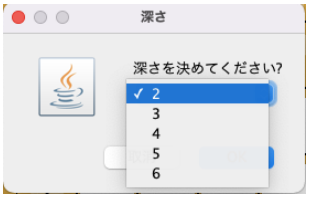

図 5　深さ選択

3.2 αβ法

今回の探索に αβ 法を用いる．αβ 法は,MinMax 法と同じ結果を得られるにもかかわらず, 理論上の計算量は MinMax 法と比較した場合, 同じ時間でほぼ二倍の深さまで読めるというアルゴリズムである．αβ 法は探索範囲を減らすため「α カットと β カット」という手法を用いる．図 6, 図 7 に α カット, β カットの例を示す．図 6 において, 頂点 C の値は子の頂点 D,E,F のうち最も小さい値となる．探索により頂点 D の値が 4 だとわかった時点で C の値は 4 以下となることが確定する．一方, 頂点 A の値は, 子の頂点 B, C のう最も大きな値

となる．B の値は 10 であることが確定しており, 頂点 D の値が確定したことにより頂点 C の値が B の値を上回らない事がわかったので頂点 E,F 以下は探索しない．これが α カットである．同様に図 7 において, 頂点 J の値が 10 だとわかった時点で I の値が 10 以上になることが確定する．これにより頂点 I の値が頂点 H の値を下回らない事がわかったので頂点 K,L 以下は探索しない．これが β カットである．このように α カット, β カットは探索する必要がない枝を探索しないための工夫である．

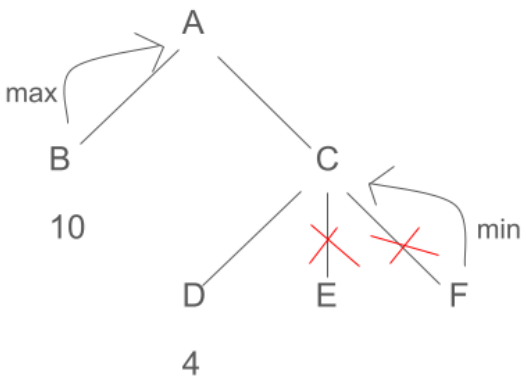

図 6　α カット

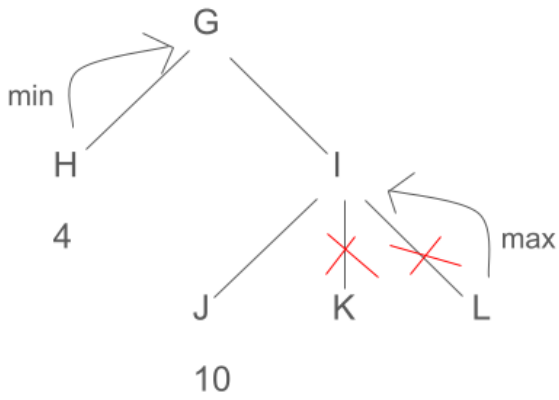

図 7　β カット

3.3　詰将棋ルーチン

本研究では詰将棋ルーチンを使い処理時間の短縮を図る．着手選択において, 可能な手の中に王手を掛ける手がある場合, まず詰将棋ルーチンを行う．詰め将棋ルーチンでは, 攻め手は着手可能手のうち王手になる手のみを選択し, 受け手は王手から逃れる手のみを選択する．このとき, 攻め手の着手可能手の中に王手となる手が一つも無ければそこで探索は終了する．選択肢を減らすことにより詰将棋ルーチンは通常の探索より短時間でできることが期待できる．詰め将棋ルーチンにより一定手数先までに詰みがなければ, その後改めて通常探索を行う．

図 8 に詰将棋ルーチンの例を示す．図 8 において, 右端の手は王手を掛ける手である．そこで詰め将棋ルーチンを発動し, 以降では攻め手は王手となる手のみを, 受け手は王手から逃れる手のみを探索する．

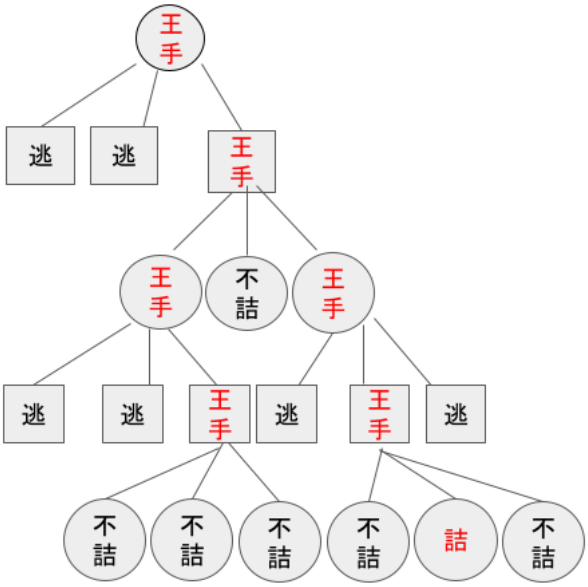

図 8　詰将棋ルーチン

3.4　プログラムの構造

本節では，本研究で作成した 5 五将棋 AI のプログラムの構造について述べる．付録 1 に本研究で作成したプログラムのソースプログラムを示す．

本研究で作成したプログラムは以下のパッケージとクラスから成る．

- board パッケージ
  - Ban クラス
  - Baord クラス
  - Calc_kyokumen_foul クラス
  - Calc_move_human クラス
  - Free_kyokumen クラス
  - Komadai クラス
  - Kyokumen クラス
  - Make_board_list クラス
  - Move クラス
  - ValueComparetor_gote クラス
  - ValueComparetor_sente クラス
- caclValue パッケージ
  - Foul_value クラス
  - Likely_to_be_token_value クラス
  - Loss_koma_value クラス
  - Move_place_value クラス

  - Parent_value クラス
  - Safety_gyoku_value クラス
  - Tada_value クラス
  - Tumi_check クラス
  - Value クラス
- koma パッケージ
  - Calc_Koma クラス
  - Gin クラス
  - Gyoku クラス
  - Hisha クラス
  - Hu クラス
  - Kaku クラス
  - Kin クラス
  - Koma クラス
  - Narigin クラス
  - Ryu クラス
  - To クラス
  - Uma クラス
- main パッケージ
  - Calc クラス
  - Kihu クラス
  - Main_swing クラス
  - Main クラス
  - Player クラス
- swing パッケージ
  - Action_kihu_listener クラス
  - Action_listener_menu クラス
  - Action_listener クラス
  - Ban_panel クラス
  - Frame_s クラス
  - ImageIcon2 クラス
  - Kihu_panel クラス
  - Komadai_panel クラス
  - Make_koma_Image クラス
  - Menu_panel クラス
  - State_game クラス

次節以降，各パッケージと各クラスについて述べる．

3.5 board パッケージ

3.5.1 Ban クラス

Ban クラスは盤を生成して駒を配置するクラスである. 表 1 に Ban クラスのクラス図を示す.

表 1 Ban クラスのクラス図

Ban	
-banarray : Koma[][]	盤面
+Ban()	コンストラクタ
+Ban(Koma[][] banarray)	コンストラクタ
+clone() : clone	クローン生成
+equal(Ban ban) : boolean	同値判定
+save_file(FileWriter file_writer) : void	ファイルへの書き出し
+load_file(BufferedReader br) : boolean	ファイルからの読み込み
+output_ban() : void	盤面出力
+set_first_ban() : void	初期局面生成
+set_banarray(int a,int b,Koma koma) : void	盤に駒をセット
+get_banarray(int a,int b) : Koma	指定した座標の駒を得る

3.5.2 Board クラス

Board クラスは探索方法や, 探索の選択など画面に関するクラスである. 表 2 に Board クラスのクラス図を示す.

表 2: Board クラスのクラス図

Board	
-kyokumen : Kyokumen	局面
-player : Player	プレイヤー
-before_move : Move	直前の動き
-board_list : ArrayList <Board >	全ての手の候補リスト
-value : Value	評価値
-max_depth : int	読む深さ
-kyokumen_list : ArrayList <Kyokumen >	千日手チェックに必要
-kihu : Kihu	棋譜を保存する
-routine : int	探索方法
-seTime : long	処理時間
-count_tumi : int	手数
+Board(Kyokumen kyokumen,Player player,ArrayList <Kyokumen >kyokumen_list,Kihu kihu)	コンストラクタ
+Board(Kyokumen kyokumen,Player player,ArrayList <Kyokumen >kyokumen_list)	コンストラクタ
+Board(Kyokumen kyokumen)	コンストラクタ
+read_kihu(String load_file) : void	棋譜読み込み
+read_kyokumen(String load_file) : void	局面読み込み
+set_first() : void	初期画面にする
+reset_except_kyokumen() : void	局面以外はリセット
-set_board(Board board) : void	引数の board に変身する
+save_kyokumen(String save_file) : void	引数の board に変身する
+check_sennitite(ArrayList <Kyokumen >kyokumen_list) : int	千日手か確認
+check_sennitite() : int	千日手か確認
+check_tumi() : boolean	積んでいるか確認
+check_foul() : boolean	反則ないか確認
+decide_all_koma_move_place() : void	全てのコマの移動場所を確認
+output_kyokumen() : void	局面を出力
+move_back_kyokumen(Move move) : void	Move をもらって戻る
+get_move_next_board(Move move) : Board	move から次の局面になる board を返す
+move_next_kyokumen(Move move) : void	move をもらって局面を進める
+go_next_board() : Board	次の局面へいく
+go_next_board_from_swing() : int	次の局面がなければ 0 返す
+sente_tumi_go_next() : Board	先手詰将棋ルーチン
+gote_tumi_go_next() : Board	後手詰将棋ルーチン
+ryoute_tumi_go_next() : Board	両手詰将棋ルーチン
+ab_go_next() : Board	通常探索
+get_seTime : long	総処理時間を返す
+get_count : int	手数を返す
+reset_count_time : void	総処理時間をリセット
+select_routine() : Board	ルーチンを選べる
-sort_board_list() : void	評価値を高い順に並べ替える
+decide_next_board_no_parent(int depth) : board	board_list 作り, 次の局面を返す
-decide_next_board_have_parent(int depth,Board board_parent) : Board	board_list 作り, 次の局面を返す
-compare_board(Board b1,Board b2) : boolean	b1 の評価値の方がよければ true
-select_board_list() : void	勝っているのがあればそれ一つにする
-make_board_list() : void	全ての手のリストを作る
+set_value_this() : void	評価値を読む
+output_value_test() : void	評価値を出力
+get_teban_s() : String	手番を返す
+get_teban_opposite_s() : String	相手の手番を返す
+get_before_move() : Move	前回の動きを返す
+get_value() : Value	評価値を返す
+set_before_move(Move move) : void	前回の動きを読む
+get_kyokumen() : Kyokumen	局面を返す
+get_board_list() : ArrayList <Board >	board_list を返す
+get_teban() : int	手番を返す
+set_player_sente(boolean sente) : void	先手のプレイヤーを読む
+set_player_gote(boolean gote) : void	後手のプレイヤーを読む
+get_player_both() : String	両方のプレイヤーを返す
+get_teban_player() : String	手番のプレイヤーを返す
+get_kyokumen_list() : ArrayList <Kyokumen >	局面のリストを返す
+get_kihu() : Kihu	棋譜を返す
+save_kihu(String file_name) : void	棋譜を保存する
+delete_last_kyokumen_list() : void	最後に保存した局面を削除する
+contain_cp() : boolean	プレイヤーに CP がいれば true
+set_kyokumen(Kyokumen kyokumen) : void	局面を読み込む
+set_kihu(Kihu kihu) : void	棋譜を読み込む
+set_depth_max : void	深さを読み込む
+set_routine : void	ルーチンを読み込む

3.5.3 Calc_kyokumen_foul クラス

Calc_kyokumen_foul クラスは千日手かどうかを調べるクラスである．表 3 に Calc_kyokumen_foul クラスのクラス図を示す

表 3　Calc_kyokumen_foul クラスのクラス図

Calc_kyokumen_foul	
+check_sennitite(ArrayList <Kyokumen >kyokumen _list,Kyokumen kyokumen) : int	千日手かを調べる
+check_foul_all(Kyokumen kyokumen,Move before_move) : boolean	今の局面に反則がないか確認する
+check_utihuzume(Kyokumen kyokumen,Move before_move) : boolean	今の局面が打ち歩詰めになっているか確認
+check_oute(Kyokumen kyokumen,int teban) : boolean	手番に王手がかかっているか確認
-check_oute_ignore(Kyokumen kyokumen) : boolean	王手放置しているか確認
-check_koma_number(Kyokumen kyokumen) : boolean	コマの数を数える
-check_dontmove(Kyokumen kyokumen) : boolean	動かせない駒 (歩、香、桂) があるか確認
-check_nihu(Kyokumen kyokumen) : boolean	二歩があるか確認

3.5.4 Calc_move_human クラス

Calc_move_human クラスは人間の入力に関するクラスである．表 4 に Calc_move_human クラスのクラス図を示す

表 4　Calc_move_human クラスのクラス図

Calc_move_human	
-kyokumen : Kyokuen	局面
+Calc_move_human(Kyokumen kyokumen)	コンストラクタ
+get_move() : Move	正しい入力の move を返す
-check_go_next_kyokumen(Move move) : boolean	次の局面にいけるか確認する
-check_go_before_place(int before_a,int before_b) : boolean	before_a,before_b の場所に駒があるか確認する
-make_move() : Move	move クラスを作る
-get_teban() : int	手番を返す

3.5.5 Free_kyokumen クラス

Free_kyokumen クラスは局面を自由に編集できるクラスである．表 5 に Free_kyokumen クラスのクラス図を示す．

表 5　Free_kyokumen クラスのクラス図

Free_kyokumen	
+Free_kyokumen(Kyokumen kyokumen)	コンストラクタ
+get_kyokumen(int teban) : Kyokumen	局面を返す
+move(int before_place,int after_place) : Koma	駒を動かす

3.5.6 Komadai クラス

Komadai クラスは持ち駒を管理する駒台のクラスである．表 6 に Komadai クラスのクラス図を示す．

表 6　Komadai クラスのクラス図

Komadai	
-komadai[] : int	駒台のリスト
-teban : int	手番
+Komadai(int[] komadai,int teban)	コンストラクタ
+Komadai(int teban)	コンストラクタ
+clone() : Komadai	クローン生成
+equal(Komadai komadai_n) : boolean	同値判定
+save_file(FileWriter file_writer) : void	ファイルに保存する
+load_file(BufferedReader br) : boolean	正しく読み込めたら true
+get_all_point_teban() : int	駒台にある駒のポイントの合計を返す
+get_komadai_all_koma() : ArrayList <Koma >	駒台にある全ての駒のリストを返す
+output_komadai() : void	駒台のコマを出力
+decrease_komadai(Koma koma) : void	駒を打つ
+decrease_komadai_koma_number(int koma_number) : void	駒を打つ
+set_komadai(Koma koma) : void	駒を駒台に入れる
+get_komadai(int place) : Koma	駒台を返す
+get_koma_number(int place) : int	駒の数を返す
+set_teban(int teban) : void	手番を読み込む
+get_teban() : int	手番を返す
-get_teban_s() : String	String で手番を返す
-change_n_s(int n) : String	n をそれぞれの駒に変える

3.5.7 Make_board_list クラス

Make_board_list クラスは Board のリストを作成するクラスである．表 7 に Make_board_list クラスのクラス図を示す．

表 7　Make_board_list クラスのクラス図

Make_board_list	
+get_next_move_list_only_oute(Kyokumen kyokumen) : ArrayList <Move>	次の手で王手のリストを作る
+get_next_move_list_no_foul(Kyokumen kyokumen) : ArrayList <Move>	kyokumen クラスから次に行ける Move クラスのリストを返す
+get_next_board_list(Board board) : ArrayList <Board>	次の手の Board のリストを返す
-make_move_list(Kyokumen kyokumen,int teban) : ArrayList <Move>	kyokumen から全ての駒の全ての動き方のリストを作る

3.5.8 Kyokumen クラス

Kyokumen クラスは局面に関するクラスである. 表 8 に Kyokumen クラスのクラス図を示す.

表 8: Kyokumen クラスのクラス図

Kyokumen	
-sente_komadai : Komadai	先手の駒台
-gote_komadai : Komadai	後手の駒台
-ban : Ban	盤
-teban : int	手番
-koma_array : ArrayList <Koma >	全ての駒のリスト
+Kyokumen(Ban ban,Komadai sente_komadai, Komadai gote_komadai,int teban)	コンストラクタ
+Kyokumen()	コンストラクタ
+clone() : Kyokumen	クローン生成
+equal(Kyokumen kyokumen) : boolean	局面を比較する
-clone_all_koma : void	全ての駒のクローン作って置き換える
+get_teban_koma_list_ban(int teban) : ArrayList <Koma >	自分の駒のリストを返す (盤上のみ)
+get_teban_koma_list_all(int teban) : ArrayList <Koma >	自分の駒のリストを返す (持ち駒も含む)
+get_place_gyoku(int teban) : int	手番の王様の場所を返す
+decrease_motigoma(Koma koma) : void	打った駒を受け取って駒台のその駒を減らす
+move_back_kyokumen(Move move) : void	move もらって前の局面へ
+move_next_kyokumen(Move move) : void	move もらって次の局面へ
-change_teban() : void	手番を変える
+decide_all_koma_move_place() : void	全ての駒の置く場所を決める
+add_koma_array(Koma koma) : void	引数の駒をリストに加える
+remove_koma_array(Koma koma) : void	引数の駒をリストから削除する
+set_koma_array() : void	駒のリストを作る
+output_kyokumen() : void	局面を出力する
+save_kyokumen_from_file_writer(FileWriter file_writer) : void	file_writer を受け取って局面をそのファイルに保存する
+save_kyokumen(String save_file) : void	局面をファイルに保存する
+read_kyokumen_from_first_kyokumen(BufferedReader br) : boolean	局面を読み込む
+load_kyokumen(String load_file) : boolean	局面を読み込む
+output_koma_array() : void	局面を出力する
+get_koma_from_place(int place_a,int place_b) : Koma	入力の場所の駒を返す
+get_koma_from_place(int place) : Koma	入力の場所の駒を返す
+set_koma_from_place(Koma koma,int place_a,int place_b) : void	場所に駒をセットする
+get_sente_komadai_koma(int place_b) : Koma	先手の駒台の駒を返す
+get_gote_komadai_koma(int place_b) : Koma	後手の駒台の駒を返す
+get_sente_komadai() : Komadai	先手の駒台を返す
+get_gote_komadai() : Komadai	後手の駒台を返す
+get_komadai(int teban) : Komadai	駒台を返す
+get_banarray(int a,int b) : Koma	指定した座標の駒を返す
+get_banarray(int place) : Koma	指定した座標の駒を返す
+get_teban() : int	手番を返す
+get_koma_array() : ArrayList <Koma >	駒のリストを返す
+get_ban() : Ban	盤を返す
+get_teban_string() : String	
+set_ban_komadai(Ban ban,Komadai sente, Komadai gote) : void	盤と駒台をセットする

3.5.9 Move クラス

Move クラスは駒の移動に関するクラスである. 表 9 に Move クラスのクラス図を示す.

表 9　Move クラスのクラス図

Move	
-before_place_a : int	1 一の 1 , 駒台なら 10,20
-before_place_b : int	1 一の一
-after_place_a : int	移動後の座標 A
-after_place_b : int	移動後の座標 B
-naru : boolean	true で成る
-get_koma :int	取った駒, 取っていなければ 0
+Move()	コンストラクタ
+Move(Move move)	コンストラクタ
+Move(int before_a,int before_b,int after_a,int after_b)	コンストラクタ
+clone() : Move	クローン生成
+equals_move(Move move) : boolean	move が等しいか判定
+check_naru(Koma koma) : boolean	駒が成れるか判断
+able_naru(int teban) : boolean	駒が成れるか判断
+check_able_naru(Koma koma) : boolean	駒が成れるか判断
+output_move_test() : void	move の出力
+output_move_kihu() : void	棋譜を保存するときのアウトプット
+get_move_kihu() : String	棋譜を保存するときの改行を含む文字列を返す
+input_move() : void	move の入力
+get_input_move(String str) : Move	新しく move クラス作って返す
+match_after_place(int a,int b) : boolean	入力と同じ場所が after_place か
+match_after_place(int place) : boolean	入力と同じ場所が after_place か
+get_before_place_a() : int	before_place_a を返す
+get_before_place_b() : int	before_place_b を返す
+get_after_place_a() : int	after_place_a を返す
+get_after_place_b() : int	after_place_b を返す
+set_before_place_a(int before_a) : void	before_a をセットする
+set_before_place_b(int before_b) : void	before_b をセットする
+set_place(int before_a,int before_b,int after) : void	それぞれの place にセットする
+set_after_place_a(int after_a) : void	after_a をセットする
+set_after_place_b(int after_b) : void	after_b をセットする
+set_after_place(int after_place) : void	それぞれの place にセットする
+get_naru() : boolean	成りの状態を返す
+get_get_koma() : int	get_koma を返す
+set_naru(boolean naru) : void	成りの状態をセットする
+set_get_koma(int koma) : void	get_koma にセットする
+get_after_place_mix() : int	after_place を合わせたものを返す

3.5.10 ValueComparator_gote クラス

ValueComparator_gote クラスは後手の情報を比べるクラスである．表 10 に ValueComparator_gote クラスのクラス図を示す．

表 10　ValueComparator_gote クラスのクラス図

ValueComparator_gote	
+compare(Board b1, Board b2)：int	b1 と b2 の評価値の比較
+compare_board_gote(Board b1,Board b2)：boolean	b1 が小さければ true

3.5.11 ValueComparator_sente クラス

ValueComparator_sente クラスは先手の情報を比べるクラスである．表 11 に ValueComparator_sente クラスのクラス図を示す．

表 11　ValueComparator_sente クラスのクラス図

ValueComparator_sente	
+compare(Board b1, Board b2)：int	b1 と b2 の評価値の比較
+compare_board_sente(Board b1,Board b2)：boolean	b1 が小さければ true

3.6 calcValue パッケージ

3.6.1 Foul_value クラス

Foul_value クラスは反則を選ばないようにするためのクラス．表 12 に calcValue クラスのクラス図を示す．

表 12　Foul_value クラスのクラス図

Foul_value	
+Foul_value(Kyokumen kyokumen)	コンストラクタ
+calc_value(Move before_move)：void	評価値を決める

3.6.2 Likely_to_be_token_value クラス

Likely_to_be_token_value クラスは自分の駒の動けるところに相手の駒があるか確認し, その評価値を決めるクラスである．表 13 に Likely_to_be_token_value クラスのクラス図を示す．

表 13　Likely_to_be_token_value クラスのクラス図

Likely_to_be_token_value	
+Likely_to_be_token_value(Kyokumen kyokumen)	コンストラクタ
+calc_value_l() : void	評価値を決める

3.6.3　Loss_koma_value クラス

Loss_koma_value クラスは駒を失う評価値を決めるクラスである．表 14 に Loss_koma_value クラスのクラス図を示す．

表 14　Loss_koma_value クラスのクラス図

Loss_koma_value	
+Loss_koma_value(Kyokumen kyokumen)	コンストラクタ
+calc_value() :void	評価値を決める

3.6.4　Move_place_value クラス

Move_place_value クラスは移動する際の評価値のクラスである．表 15 に Move_place_value クラスのクラス図を示す．

表 15　Move_place_value クラスのクラス図

Move_place_value	
+Move_place_value(Kyokumen kyokumen)	コンストラクタ
+calc_value() :void	評価値を決める

3.6.5　Parent_value クラス

Parent_value クラスは Kyokumen クラスの評価値を決める親クラスである．表 16 に Parent_value クラスのクラス図を示す．

表 16　Parent_value クラスのクラス図

Parent_value	
-kyokumen : Kyokumen	局面
-value_boolean : boolean	勝敗が決まれば true
-value_int : int	評価値の数値
+calc_value() :void	評価値を決める
+get_value_boolean() : boolean	これしかないなどの手がある時 true
+get_value_int() : int	int 型で評価値を返す
+get_kyokumen() : Kyokumen	局面を返す
+get_teban() : int	手番を返す
+set_value_int(int value_int) : void	評価値を value_int にセットする
+set_value_boolean(boolean value_boolean) : void	value_boolean に引数をセットする

3.6.6　Safely_gyoku_value クラス

Safely_gyoku_value クラスは王の安全度を評価するクラスである．表 17 に Safely_gyoku_value クラスのクラス図を示す．

表 17　Safely_gyoku_value クラスのクラス図

Safely_gyoku_value	
+Safely_gyoku_value(Kyokumen kyokumen)	コンストラクタ
+calc_value() :void	評価値を決める
-make_value(int teban) : int	teban の玉の危険度を返す
-check_diff(int diff) : boolean	diff が 1 マス以内か判断

3.6.7　Tada_value クラス

Tada_value クラスはただで取れる駒の評価値を確認するクラスである．表 18 に Tada_value クラスのクラス図を示す．

表 18　Tada_value クラスのクラス図

Tada_value	
+Tada_value(Kyokumen kyokumen)	コンストラクタ
+calc_value() :void	評価値を決める
-check_enemy_koma_move(Move move) : boolean	相手の駒が move の移動先に効いているか確認する

3.6.8 Tumi_check クラス

Tumi_check クラスは詰んでいるかを確認するクラスである．表 19 に Tumi_check クラスのクラス図を示す．

表 19　Tumi_check クラスのクラス図

Tumi_check	
+check_tumi_teban(Kyokumen kyokumen) : boolean	手番の玉がその時詰んでいるかを確認する
+check_tumi_n_te(Kyokumen kyokumen,int n) : Move	局面で n 手以下の詰みがあるか調べる
+check_tumi_for_gyoku(Kyokumen kyokumen,int n) : boolean	王手されている局面で n 手以下で詰まされるか調べる

3.6.9 Value クラス

Value クラスは評価値を計算するクラスである．表 20 に Value クラスのクラス図を示す．

表 20: Value クラスのクラス図

Value	
- value : int	評価値
-determe : boolean	勝負が決まっている時 true
+Value()	コンストラクタ
+calc_value_present(Kyokumen kyokumen,ArrayList <Kyokumen>kyokumen_list,Move before_move) :void	評価値を計算する
-add_value(int add) : void	評価値を追加する
+get_value() : int	評価値を返す
+get_determe() : boolean	determe を返す
+set_value(int value) : void	評価値をセットする
+set_determe(boolean determe) : void	determe をセットする
+set_value_all(Value value) : void	評価値と determe をセットする
+set_win(int teban) : void	勝ちが決まっている時のその設定をする
-set_lose(int teban) : void	負けが決まっている時にその設定をする
+get_win(int teban) : boolean	teban が勝っていれば true

3.7 koma パッケージ

3.7.1 Calc_Koma クラス

Calc_Koma クラスは駒を作るクラスである．表 21 に Calc_Koma クラスのクラス図を示す．

表 21　Calc_Koma クラスのクラス図

Calc_Koma	
+make_load_Koma(String load_str) : Koma	ファイルから局面を読み込んだ時の読み込み
-change_teban_int_from_string(String str) : int	String 型を int 型に変える
+make_koma(int n,int teban) : Koma	番号から駒を作る
+isBan(int a,int b) : boolean	a*10+b が 11-99 の中にあるなら true
+isBan(int place) : boolean	a*10+b が 11-99 の中にあるなら true

3.7.2 Koma クラス

Koma クラスは駒を定義するクラスである. 表 22 に Koma クラスのクラス図を示す.

表 22: Koma クラスのクラス図

Koma	
-teban : int	手番
-place_a : int	1 一の 1
-place_b : int	1 一の一
-point : int	駒の価値を表すポイント
-point_special : int	特別なポイント
-move_place : ArrayList<Integer>	動ける場所
-motigoma : boolean	持ち駒なら true
+Koma(int teban_n)	コンストラクタ
+clone() : Koma	クローン生成
+equal(Koma koma) : boolean	同値生成
+save_file(FileWriter file_writer) : void	ファイルを保存する
+able_to_move() : boolean	動けるか, 反則にならないか
+get_narazukoma() : Koma	不成の駒作る
+get_narigoma() : Koma	成り駒を返す
+able_to_naru() : boolean	成れる駒は true を返す
+get_koma_name() : String	駒の名前を全角 2 文字で返す
-get_teban_s() : String	手番を全角文字で返す
+output_koma() : void	全角 2, 半角 1, ↑↓, 半角スペース 1 で出力する
+output_test() : void	テスト用のアウトプット
+move_place_clear() : void	move_place を一つ削除する
+decide_move_place(Kyokumen kyokumen) : void	動ける場所を決める
+move_place_more(Kyokumen kyokumen,int dif_a,int dif_b) : void	香車などずっと動ける奴の動き
+contain_move_place(int place) : boolean	place が move_place に含まれるか
+output_move_place() : void	自分のいる場所、駒の名前、動ける場所を出力
+set_teban(int teban_n) : void	手番をセットする
+get_teban() : int	手番を返す
+set_place(int a,int b) : void	引数の場所をセットする
+set_point(int point_n) : void	引数のポイントをセットする
+get_point() : int	ポイントを返す
+set_point_special(int point_special_n) : void	引数の特別なポイントをセットする
+get_point_special() : int	特別なポイントを返す
+set_move_place(ArrayList¡Integer¿ move_place) : void	移動可能な場所をセットする
+get_move_place() : ArrayList<Integer>	移動可能な場所を返す
+get_move_place_clone_deep() : ArrayList<Integer>	move_place のクローン生成
+add_move_place(int place) : void	移動可能な場所を追加する
+get_place_a() : int	place_a を返す
+get_place_b() : int	place_b を返す
+get_koma_number() : int	駒の番号を返す
+set_motigoma(boolean boo) : void	持ち駒をセットする
+get_motigoma() : boolean	持ち駒なら true
+get_place() : int	place_a と place_b を合わせたものを返す
+get_point_teban() : int	手番のポイントを返す
+change_teban() : void	手番を変える

3.7.3 Hu クラス

Hu クラスは歩の駒のクラスである. 表 23 に Hu クラスのクラス図を示す.

表 23　Hu クラスのクラス図

Hu	
+Hu(int teban_n)	コンストラクタ
+able_to_move() : boolean	動けるか, 反則にならないか
+able_to_naru() : boolean	なれる駒は true を返すようにする
+get_koma_number() : int	駒の番号を返す, 歩は 1
+get_koma_name() : String	駒の名前を返す
+decide_move_place(Kyokumen kyokumen) : void	動ける場所を決める

3.7.4 Kin クラス

Kin クラスは金の駒のクラスである．表 24 に Kin クラスのクラス図を示す．

表 24　Kin クラスのクラス図

Kin	
+Kin(int teban_n)	コンストラクタ
+get_koma_number()：int	駒の番号を返す, 金は 3
+get_koma_name()：String	駒の名前を返す
+decide_move_place(Kyokumen kyokumen)：void	動ける場所を決める

3.7.5 Gin クラス

Gin クラスは銀の駒のクラスである．表 25 に Gin クラスのクラス図を示す．

表 25　Gin クラスのクラス図

Gin	
+Gin(int teban_n)	コンストラクタ
+able_to_naru()：boolean	なれる駒は true を返すようにする
+get_koma_number()：int	駒の番号を返す, 銀は 2
+get_koma_name()：String	駒の名前を返す
+decide_move_place(Kyokumen kyokumen)：void	動ける場所を決める

3.7.6 Gyoku クラス

Gyoku クラスは玉の駒のクラスである．表 26 に Gyoku クラスのクラス図を示す．

表 26　Gyoku クラスのクラス図

Gyoku	
+Gyoku(int teban_n)	コンストラクタ
+get_koma_number()：int	駒の番号を返す, 玉は 4
+get_koma_name()：String	駒の名前を返す
+decide_move_place(Kyokumen kyokumen)：void	動ける場所を決める

3.7.7 Kaku クラス

Kaku クラスは角の駒のクラスである．表 27 に Kaku クラスのクラス図を示す．

表 27　Kaku クラスのクラス図

Kaku	
+Kaku(int teban_n)	コンストラクタ
+able_to_naru() : boolean	なれる駒は true を返すようにする
+get_koma_number() : int	駒の番号を返す, 角は 5
+get_koma_name() : String	駒の名前を返す
+decide_move_place(Kyokumen kyokumen) : void	動ける場所を決める

3.7.8　Hisha クラス

Hisha クラスは飛車の駒のクラスである. 表 28 に Hisha クラスのクラス図を示す.

表 28　Hisha クラスのクラス図

Hisha	
+Hisha(int teban_n)	コンストラクタ
+able_to_naru() : boolean	なれる駒は true を返すようにする
+get_koma_number() : int	駒の番号を返す, 飛車は 6
+get_koma_name() : String	駒の名前を返す
+decide_move_place(Kyokumen kyokumen) : void	動ける場所を決める

3.7.9　To クラス

To クラスはとの駒のクラスである. 表 29 に To クラスのクラス図を示す.

表 29　To クラスのクラス図

To	
+To(int teban_n)	コンストラクタ
+get_koma_number() : int	駒の番号を返す, とは 11
+get_koma_name() : String	駒の名前を返す

3.7.10　Narigin クラス

Narigin クラスは成銀の駒のクラスである. 表 30 に Narigin クラスのクラス図を示す.

表 30　Narigin クラスのクラス図

Narigin	
+Narigin(int teban_n)	コンストラクタ
+get_koma_number() : int	駒の番号を返す, 成銀は 12
+get_koma_name() : String	駒の名前を返す

3.7.11　Ryu クラス

Ryu クラスは龍の駒のクラスである. 表 31 に Ryu クラスのクラス図を示す.

表 31　Ryu クラスのクラス図

Ryu	
+Ryu(int teban_n)	コンストラクタ
+get_koma_number() : int	駒の番号を返す, 龍は 16
+get_koma_name() : String	駒の名前を返す
+decide_move_place(Kyokumen kyokumen) : void	動ける場所を決める

3.7.12　Uma クラス

Uma クラスは馬の駒のクラスである. 表 32 に Uma クラスのクラス図を示す.

表 32　Uma クラスのクラス図

Uma	
+Uma(int teban_n)	コンストラクタ
+get_koma_number() : int	駒の番号を返す, 馬は 15
+get_koma_name() : String	駒の名前を返す
+decide_move_place(Kyokumen kyokumen) : void	動ける場所を決める

3.8　main パッケージ

3.8.1　Calc クラス

Calc クラスは文字列が数字か確認するクラスである. 表 33 に Calc クラスのクラス図を示す.

表 33　Calc クラスのクラス図

Calc	
+isNumber(String str) : boolean	文字列が数字かどうかを判断
+change_teban(int teban) : int	手番を変える

3.8.2　Kihu クラス

Kihu クラスは棋譜を保存するクラスである．表 34 に Kihu クラスのクラス図を示す．

表 34　Kihu クラスのクラス図

Kihu	
-kihu_list : ArrayList<Move>	棋譜のリスト
-first_kyokumen : Kyokumen	初期の局面
-last_kyokumen : Kyokumen	対戦終了時の局面
+Kihu()	コンストラクタ
+Kihu(Kyokumen kyokumen)	コンストラクタ
+Kihu(ArrayList<Move>kihu_list,Kyokumen first_kyokumen)	コンストラクタ
+clone_kihu() : Kihu	クローン生成
+make_last_kyokumen() : void	対戦終了時の局面を作る
+read_kihu_from_first_kyokumen(String input_file) : void	最初の局面から保存されている棋譜を読み込む
+input_kihu(String input_file) : void	kihu ファイルを読み込む
+save_kihu_from_first_kyokumen(String save_file) : void	棋譜ファイルを最初の局面から保存する
+save_kihu_file(String save_file) : void	棋譜をファイルに保存する
+output_kihu() : void	棋譜を出力
+add_move(Move move) : void	kihu_list に駒の移動を追加
+get_kihu_list() : ArrayList<Move>	kihu_list を返す
+get_last_move() : Move	最後の駒の移動を返す
+remove_last_move() : void	最後の駒の移動を削除
+get_first_kyokumen() : Kyokumen	初期局面を返す
+get_last_kyokumen() : Kyokumen	最後の局面を返す

3.8.3　Main_swing クラス

MainSwing クラスは Main クラスから画面を呼び出す実行クラスである．

3.8.4　Main クラス

Main クラスは対局を管理する実行クラスである．

3.8.5 Player クラス

Player クラスは先手後手が人間か CP かを確認するクラスである．表 35 に Player クラスのクラス図を示す.

表 35　Player クラスのクラス図

Player	
-player_sente : boolean	先手が人間か CP か
-player_gote : boolean	後手が人間か CP か
+Player()	コンストラクタ
+decide_player() : void	先手と後手のプレイヤー決める
+output_player() : void	プレイヤーを出力する
+get_player_both() : String	両プレイヤーの種類を返す
-get_player_sente_string() : String	先手プレイヤーの種類を返す
-get_player_gote_string() :String	後手プレイヤーの種類を返す
+get_player_sente() : boolean	先手プレイヤーの種類を返す.CP なら true
+get_player_gote() : boolean	後手プレイヤーの種類を返す.CP なら true
+get_player(int teban) : boolean	引数の手番のプレイヤーの種類を返す.CP なら true
+set_player_sente(boolean player_sente) : void	先手プレイヤーの種類をセットする
+set_player_gote(boolean player_gote) : void	後手プレイヤーの種類をセットする
+contain_cp() : boolean	cp が含まれていれば true

3.9 swing パッケージ

3.9.1 Action_kihu_listener クラス

Action_kihu_listener クラスはアクションイベントの棋譜に関するクラスである． 表 36 に Action_kihu_listener クラスのクラス図を示す.

表 36　Action_kihu_listener クラスのクラス図

Action_kihu_listener	
-board : Board	盤面
-frame : Frame_s	設定に関するパネル
-state : State_game	ゲームの状態,1 なら対局中
+Action_kihu_listener(Board board,Frame_s frame,State_game state)	コンストラクタ
+actionPerformed(ActionEvent e) : void	アクションイベント

3.9.2 Action_listener_menu クラス

Action_listener_menu クラスはアクションイベントのメニューのクラスである. 表 37 に Action_listener_menu クラスのクラス図を示す.

表 37 Action_listener_menu クラスのクラス図

Action_listener_menu	
-board : Board	盤面
-frame : Frame_s	設定に関するパネル
-state : State_game	ゲームの状態,1 なら対局中
-save_kyokumen_name : String	局面を保存する時のファイル名
-read_kyokumen_name : String	局面を読み込む時のファイル名
-save_kihu_name : String	棋譜を保存するときのファイル名
-read_kihu_name : String	棋譜を読み込むときのファイル名
+depth : int	探索する深さ
+routine : int	探索方法
+Action_listener_menu(Board board,Frame_s frame,State_game state)	コンストラクタ
+actionPerformed(ActionEvent e) : void	アクションイベントを起こす
-ask(String question,String title) : int	質問をして yes なら 1,no なら 2
-matta() : void	局面を 1 手前に戻す
-set_this_kyokumen() : void	保存していた局面をセットする
-set_this_kihu() : void	保存していた棋譜をセットする
-set_first_kyokumen() : void	初期局面をセットする
-set_player() : boolean	先後を決める
+get_dep() : int	深さを決める
+get_go_next() : int	探索を決める
+get_dep_tumi() : int	詰ルーチンの場合の深さを決める
+get_routine() : int	ルーチンを返す

3.9.3 Action_listener クラス

Action_listener クラスはアクションイベントを受け取るためのクラスである. 表 38 に Action_listener クラスのクラス図を示す.

表 38　Action_listener クラスのクラス図

Action_listener	
-board : Board	盤面
-frame : Frame_s	設定に関するパネル
-state : State_game	ゲームの状態,1 なら対局中
-free_kyokumen : Free_kyokumen	自由に局面を変える
+Action_listener(Board board,Frame_s frame,State_game state)	コンストラクタ
+actionPerformed(ActionEvent e) : void	アクションイベント
-check_free_after(String place_s) : boolean	place が正しいかどうかを判断する
-check_free_before(String place_s) : boolean	place が正しいかどうかを判断する
-make_move(int before_place,int after_place) : Move	Move クラスを作る
-check_after_place(String place_s) : boolean	駒を選択した後に動かせる場所かどうかを判断する
-check_before_koma(String place_s) : boolean	駒が選択する前に選択できる駒かどうかを判断する
+set_free_kyokumen(Free_kyokumen free_kyokumen) : void	自由に作った盤をセットする
+get_free_kyokumen() : Free_kyokumen	自由に作った盤を返す

3.9.4　Ban_panel クラス

Ban_panel クラスは盤のパネルを管理するクラスである．表 39 に Ban_panel クラスのクラス図を示す．

表 39　Ban_panel クラスのクラス図

Ban_panel	
-grid_bag_layout : GridBagLayout	レイアウト
-panel_array : JPanel[][]	パネルのリスト
-him : Window	ウィンドウ
+Ban_panel(Window her)	コンストラクタ
+change_panel(int a,int b,Kyokumen kyokumen,Action_listener action_listener) : void	パネルの変更
+change_color(int a,int b) : void	色の変更
+set_ban_panel() : void	盤面を読み込む
-make_masu(GridBagLayout gbl) : void	駒を置くマスを作成する
+make_masu_from_ban(Ban ban,Action_listener action_listener) : void	駒を置くマスを作成する

3.9.5　Frame_s クラス

Frame_s クラスは盤や駒などのパネルの設定を行うクラスである．表 40 に Frame_s クラスのクラス図を示す．

表 40: Frame_s クラスのクラス図

Frame_s	
-panel_menu : Menu_panel	メニューパネル
-panel_ban : Ban_panel	盤のパネル
-panel_sente : Komadai_panel	先手の駒台パネル
-panel_gote : Komadai_panel	後手の駒台パネル
-label_teban : JLabel	手番を表示
-action_listener : Action_listener	アクションイベント
-action_listener_menu : Action_listener_menu	メニューのアクションイベント
-action_kihu_listener : Action_kihu_listener	棋譜のアクションイベント
-panel_kihu : Kihu_panel	棋譜パネル
-label_com : JLabel	探索方法表示
-label_dep : JLabel	深さ表示
+set_action_listener(Action_listener action_listener, Action_listener_menu action_listener_menu, Action_kihu_listener action_kihu_listener) : void	アクションをセットする
+set_next_te(String teban) : void	局面が変わる際に手番のプレイヤーをセットする
+change_label_teban(String string) : void	手番のラベルを変更する
+change_label_com(String string) : void	毎局探索方法を変更する
+change_label_dep(String string) : void	毎局深さを変更する
+check_next_or_exit() : boolean	cp 同士の対局の時に次の手に行くかどうかを確認する
+output_message(String mes,String title) : void	メッセージを出力する
+output_touryou(String teban) : void	勝敗・処理時間・手数・平均処理時間も表示
+change_panel(int place_a,int place_b,Kyokumen kyokumen) : void	panel の指定したところだけ変える
+change_button_color(int place) : void	パネルの変更
+start_game() : void	対局開始の処理
+end_game() : void	対局終了の処理
+set_free_before() : void	自由に作った局面をセット
+set_reading_kihu() : void	読み込んだ棋譜をセット
+swing2() : void	５五将棋の画面を表示
+make_panel(JPanel panel) : void	パネルを作成する
+set_kyokumen(Kyokumen kyokumen) : void	入力の局面を表示する
+set_kihu_panel_visible() : void	棋譜パネルを表示する
+set_kihu_panel_not_visible() : void	棋譜パネルを削除する
+get_action_listener() : Action_listener	アクションを返す
+set_com(String board) : void	探索方法をセットする
+set_dep(String board) : void	深さをセットする

3.9.6 ImageIcon2 クラス

ImageIcon2 クラスはアイコンの画像を管理するクラスである．表 41 に ImageIcon2 クラスのクラス図を示す．

表 41　ImageIcon2 クラスのクラス図

ImageIcon2	
+ImageIcon2(String f, Window own)	コンストラクタ

3.9.7 Kihu_panel クラス

Kihu_panel クラスは棋譜のパネルを管理するクラスである．表 42 に Kihu_panel クラスのクラス図を示す．

表 42　Kihu_panel クラスのクラス図

Kihu_panel	
-button_next : JButton	棋譜を 1 つ進めるボタン
-button_back : JButton	棋譜を 1 つ戻すボタン
-button_first : JButton	最初の棋譜に戻すボタン
-button_end : JButton	最後の棋譜まで戻すボタン
+set_kihu_panel(Action_kihu_listener action_kihu_listener) :void	棋譜パネルを表示する
+set_visible() : void	棋譜パネルを表示する
+set_not_visible() : void	棋譜パネルを消す

3.9.8　Komadai_panel クラス

Komadai_panel クラスは駒台のパネルを管理するクラスである．表 43 に Komadai_panel クラスのクラス図を示す．

表 43　Komadai_panel クラスのクラス図

Komadai_panel	
-grid_bag_layout : GridBagLayout	レイアウト
-panel_array : JPanel[]	パネルのリスト
-him : Window	ウィンドウ
+Komadai_panel(Window her)	コンストラクタ
+change_color(int b) : void	色の変更
+change_panel(int b,Komadai komadai,Action_listener action_listener) : void	パネルの変更
+set_komadai_panel(int p_w,int p_h) : void	盤面を読み込む
-make_komadai(GridBagLayout gbl) : void	駒を置くマスを作成する
+make_komadai_panel(Komadai komadai,Action_listener action_listener) : void	駒台のパネルを作成する

3.9.9　Make_koma_Image クラス

Make_koma_Image クラスは駒の画像を読み込むクラスである．表 44 に Make_koma_Image クラスのクラス図を示す．

表 44　Make_koma_Image クラスのクラス図

Make_koma_Image	
+make_koma_image(int koma_number,int teban, Window him) : ImageIcon	駒の画像を読み込む

3.9.10 Menu_panel クラス

Menu_panel クラスは画面上部のメニューボタンを表示させるクラスである．表 45 に Menu_panel クラスのクラス図を示す．

表 45　Menu_panel クラスのクラス図

Menu_panel	
-button_new：JButton	新規対局ボタン
-button_start：JButton	現在の局面から対局開始するボタン
-button_touryou：JButton	投了するボタン
-button_read_kyokumen：JButton	局面を読み込むボタン
-button_read_kihu：JButton	棋譜を読み込むボタン
-button_calc：JButton	評価値を計算するボタン
-button_tumi：JButton	詰んでいるか確認するボタン
-button_save_kyokumen：JButton	局面を保存するボタン
-button_save_kihu：JButton	棋譜を保存するボタン
-button_matta：JButton	局面を巻き戻すボタン
-button_free：JButton	自由に局面を作成するボタン
+set_gaming()：void	ボタンを対局中の状態に変える
+set_not_gaming()：void	ボタンを対局前の状態に変える
+set_free_before()：void	ボタンを局面作成中の状態に変える
+set_menu_panel(Action_listener_menu action_listener_menu)：void	メニューパネルを作成する

3.9.11 State_game クラス

State_game クラスは対局の状態を確認するクラスである．表 46 に State_game クラスのクラス図を示す．

表 46: State_game クラスのクラス図

State_game	
-state : int	状態
-before_place : int	初期の駒の 1
-kihu : Kihu	棋譜を保存する場所
-count_tesuu : int	棋譜を読み込んだ時, 何手めか数える場所
-board : Board	盤面
-frame : Frame_s	設定に関するパネル
+State_game(Board board,Frame_s frame)	コンストラクタ
+set_not_gaming() : void	画面を対局前に戻す
+set_before_place(int before_place) : void	beforePlace を設定して手番が人間で駒を選択した状態にする
+check_gaming_player() : boolean	対局中で人間が手番の時 true
+check_chose_after_place() : boolean	対局中, 人間の手番かつ駒を選択していれば true
+check_not_gaming() : boolean	対局中かどうかを判断する
+check_nothin() : boolean	対局も何もしていないか判断する
+check_chose_before_koma() : boolean	対局中, 人間の手番かつ何もしていなければ true
+check_reading_kihu() : boolean	棋譜を読み込んでいる状態なら true
+check_free_before() : boolean	駒を自由に動かせる, かつ駒を選択していないなら true
+check_free_after() : boolean	駒を自由に動かせる, かつ駒を選択しているなら true
+get_before_place() : int	前の状態の場所に移動する
+get_state() : int	現在の状態を返す
+set_gaming(boolean player) : void	対局開始の状態にする
+set_free_before() : void	駒を自由に動かせる, かつ駒を選択していない状態にする
+set_free_after() : void	駒を自由に動かせる, かつ駒を選択している状態にする
+set_reading_kihu(int tesuu) : void	棋譜を読み込んでいる状態にする
+set_tesuu_add() : void	手数を加える
+set_tesuu_decrease() : void	手数を減らす
+set_tesuu_first() : void	最初の手数にする
+set_tesuu_end() : void	最後の手数にする
+get_kihu() : Kihu	棋譜を返す
+get_count_tesuu() : int	手数を返す
+get_move_next() : Move	棋譜を次の手数に進める
+get_move_back() : Move	棋譜を前の手数に戻す
+get_board() : Board	盤面を返す

4　駒の評価値を用いた５五将棋 AI の着手選択法

本研究で作成した AI は, 一定手数先の局面を先読みし, その局面の評価値を求め, $\alpha\beta$ 法を用いて最も評価値が高くなる局面になる手を選択する．$\alpha\beta$ 法では探索の際に, α カットと β カットという手法を用いる. これは探索する必要がない枝を探索しないための工夫である．$\alpha\beta$ 法では α 以上 β 以下の範囲を探索する．普通は $\alpha=-\infty$, $\beta=+\infty$からはじめるが, この幅を縮めることで高速に探索が行なわれる．ただし, その場合には必ずしも最善手順及び最善手順での評価値が得られるとは限らなくなるが, 返してくる値は α 以下であれば得られる評価値の上限を示す値, β 以上であれば得られる評価値の下限を示す値になる．$\alpha\beta$ 法で, 理論上の最高速度を得るには探索の順番が完全に良い評価を返す順にソートされている必要がある.

探索の高速化を図るために, 本研究では, 各着手可能手に対してその手を 1 手指した局面を生成しその評価値を求める. 次に求めた評価値をソートし, 評価値の高い手から順に $\alpha\beta$ 法で探索を行う．途中に詰みが見つかればそこで探索を終了し, その手を指す詰将棋ルーチンを用いる．詰将棋ルーチンを用いることで, 探索の高速化を図る.

5　考察

詰将棋ルーチンの有無による AI の強さおよび探索時間を検証するため, 詰将棋ルーチン有り, 詰将棋ルーチン無しの AI 同士を対戦させる．本研究では, 先手詰み, 後手詰み, 両手詰み, 両手詰み無しで, 探索する深さを変更しながら AI 同士を対戦させ勝率と処理時間を検証する.

3 手詰めルーチンは各 200 回,5 手詰めルーチンは各 40 回対戦させたときの対戦結果を次の表 47 に示す.

平均処理時間は, 深さを深くした時に差が出た．深さ 3 の時に特に差が出なかったが, 深さ 5 の時は詰将棋ルーチンを組み込むのと組み込まないでは 25% ほど処理時間の差が出る.

平均手数は, 深さが 3 と 5 では 30% ほど減り, 片方の手に詰将棋ルーチンを組み込むことで対局終了までの手数が 40% ほど減ることがわかった.

結果より, 詰将棋ルーチンは探索の高速化に有用であることがわかった.

以下では，５五将棋は先手有利であると言えるか，また詰将棋ルーチンを加えることが勝率に貢献しているか統計的に検証する.

勝率 p の勝負を N 回行ったときの標準偏差 s は

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

となる．$p = 0.5$ と仮定した場合，$N = 200, N = 40$ の場合それぞれ

$$s_{200} = \sqrt{200 * 0.5 * 0.5} = 7.07, s_{40} = \sqrt{40 * 0.5 * 0.5} = 3.16$$

となる．統計学を用いると，信頼度 95% となる区間は平均値の前後 $\pm 1.96s$ の区間であるので，勝率 50% なら勝ち数は 95% の確率で試行回数 200 回では $100 \pm 7.07 * 1.96$ 勝，すなわち 87〜113 勝 (勝率 44%〜56%), 試行回数 40 回では $20 \pm 3.16 * 1.96$ 勝，すなわち 14〜26 勝 (勝率 35%〜65%) となる.

表 47 の結果より，詰将棋ルーチンの有無に関わらず，3 手詰めルーチン，5 手詰めルーチン共に先手の勝率は上記の範囲越えていることが示される．従って，５五将棋は詰将棋ルーチンの有無に関わらず先手有利であることが示される.

次に，詰将棋ルーチンの有効性を検証する．表 47 より先手の勝率は探索の深さ 3 の場合は双方詰将棋ルーチン有り, 双方詰将棋ルーチン無しでは共に 57% である．そこで，$p = 0.57$ と仮定すると $N = 200$ の場合の標準偏差 s は

$$s_{200} = \sqrt{200 * 0.57 * 0.43} = 7.00$$

となる．勝ち数が信頼度 95% となる区間は $114 \pm 7.00 * 1.96$ 勝，すなわち 101〜127 勝 (勝率 50%〜64%) の範囲となる．表 47 において、先手詰みルーチン有り後手詰みルーチン無しの場合，および先手詰みルーチン無し後手詰みルーチン有りの場合共に先手の勝率は上記の範囲に収まっている．従って，統計上, 探索の深さが 3 の場合は詰将棋ルーチンが勝率に貢献したとは言えない．

次に, 探索の深さ 5 の場合は表 47 より双方詰将棋ルーチン有りの場合は 68%, 双方詰将棋ルーチン無しの場合は 73% である．そこで，$p = 0.70$ と仮定すると $N = 40$ の場合の標準偏差 s は

$$s_{40} = \sqrt{40 * 0.70 * 0.30} = 2.89$$

となる．勝ち数が信頼度 95% となる区間は $28 \pm 2.90 * 1.96$ 勝, すなわち 22〜32 勝 (勝率 55%〜80%) の範囲となる．表 47 において、先手詰みルーチン有り後手詰みルーチン無しの場合，および先手詰みルーチン無し後手詰みルーチン有りの場合共に先手の勝率は上記の範囲に収まっている．従って，統計上，探索の深さが 5 の場合も詰将棋ルーチンが勝率に貢献したとは言えない．

探索の深さが 3,5 の時のどちらの場合も詰将棋ルーチンが勝率に貢献したとは言えなかった．単に試行回数が少ないため統計上有意にならないだけかもしれないが, 5 × 5 の盤面,6 種類の駒の数が詰将棋ルーチンに関係あるのか調べてみる必要がある．

表 47　詰将棋ルーチンの有無と勝率

詰ルーチン		探索の	勝率		平均処理	平均手数
先手	後手	深さ	先手	後手	時間 (ms)	(手)
有	無	3	0.64	0.36	1618	34.3
		5	0.80	0.20	48030	26.1
無	有	3	0.54	0.46	1404	37.7
		5	0.65	0.35	47022	23.7
有	有	3	0.57	0.43	1665	70.9
		5	0.68	0.32	45424	39.1
無	無	3	0.57	0.43	1718	68.4
		5	0.73	0.27	60690	61.2

6　結論・今後の課題

本研究では, 5 五将棋アプリケーションを作成し, 5 五将棋 AI の探索の高速化の検証を行った．3 手詰めルーチンは各 200 回,5 手詰めルーチンは各 40 回対戦させた結果から, 探索の高速化を図ることができた．しかし, 勝率の向上はみられなかった．今後の課題としては,AI の詰将棋ルーチンを加えた探索方法の改良し探索の高速化と勝率の向上を強化することが挙げられる．

謝辞

指導を受けた教員や，本研究を完成するにあたって支援を受けた研究室の諸氏に対しお礼の言葉を，独立したページに記述する．詳しくは卒業研究担当教員の指導に従うこと．

参考文献

[1] 池泰弘：Java 将棋のアルゴリズム [改定版], 工学社 (2016).

[2] 山岡忠夫：将棋 AI で学ぶディープラーニング, マイナビ出版 (2018).

[3] 伊藤毅志，新沢剛：モンテカルロ法を用いた 5 五将棋システム，情報処理学会 研究報告 Vol. 2007-GI-18, pp.1-6 (2007)

[4] 柿木義一：５五将棋における評価関数の自動学習，第 5 回 エンターテイメント認知科学研究ステーション招待講演，電気通信大学 (2008)

[5] 伊藤毅志：5 五将棋大会の動向 (2013 年〜2014 年)，情報処理学会 研究報告 Vol.2015-GI-33, No.1, pp.1–5 (2015)

[6] 塩田雅弘，伊藤 毅志：5 五将棋における自動対戦を用いた評価関数の学習，情報処理学会 研究報告 Vol.2020-GI-44, No.3, pp.1-6 (2020)

[7] ５五将棋，SDIN 無料ゲーム，https://sdin.jp/browser/board/55shogi/

[8] 西谷昂真：将棋の駒の評価値と盤面サイズの考察，近畿大学 理工学部 情報学科 2018 年度 卒業研究報告書 (2019)

[9] Java で将棋盤作成: https://stu345.hatenablog.com/entry/2019/09/04/002035

[10] ５五将棋過去大会リンク：shttp://minerva.cs.uec.ac.jp/cgi-bin/uec55shogi/wiki.cgi?page=

[11] 伊藤琢巳，河野泰人，脊尾昌宏， 野下浩平：詰将棋を解くプログラムの進歩，人工知能学会誌 No.10, Vol.6, pp.853–859 (1995) http://id.nii.ac.jp/1004/00003896/

[12] 脊尾昌宏：C*アルゴリズムによる AND/OR 木の探索および詰将棋プログラムへの応用情報処理学会研究報告 知能と複雑系, Vol.1995, No.23, pp.103–110 (1995) http://id.nii.ac.jp/1001/00050901/

[13] 野下浩平：詰将棋を解くプログラム T2，松原仁 編著，コンピュータ将棋の進歩, 第 3 章, pp.50–70. 共立出版 (1996)

[14] 脊尾昌宏：共謀数を用いた詰将棋の解法，. 松原仁 編著，コンピュータ将棋の進歩 2, 第 1 章, pp.1–21. 共立出版 (1998)

[15] 脊尾昌宏：詰将棋を解くアルゴリズムにおける優越関係の効率的な利用について，ゲームプログラミングワークショップ 1999 論文集, pp,129-136, 情報処理学会 (1999)
http://id.nii.ac.jp/1001/00097443/

[16] 長井歩，今井浩：df-pn アルゴリズムの詰将棋を解くプログラムへの応用，情報処理学会論文誌, Vol.43, No.6, pp.1769–1777 (2002) http://id.nii.ac.jp/1001/00011597/

[17] 長井歩：df-pn アルゴリズムと詰将棋を解くプログラムへの応用，松原仁 編著，コンピュータ将棋の進歩 4, 第 5 章, pp.96–114. 共立出版 (2003)

[18] 金子知適，田中哲朗，山口和紀，川合慧：詰将棋における df-pn+ 探索のための，展開後の証明数と反証数を予測する評価関数，ゲームプログラミングワークショップ 2004 論文集，pp.14–21, 情報処理学会 (2004) http://id.nii.ac.jp/1001/00097541/

[19] 岸本章宏：詰将棋を解くための探索技術について，人口知能学会誌, No.26. Vol.4, pp.392–398 (2011) http://id.nii.ac.jp/1004/00007804/

付録 A　ソースプログラム

本研究で作成した 5 五将棋プログラムのソースを以下に示す.

・Ban クラス

```
1  package board;
2
3  import java.io.BufferedReader;
4  import java.io.FileWriter;
5  import java.io.IOException;
6
7  import koma.Calc_koma;
8  import koma.Gin;
9  import koma.Gyoku;
10 import koma.Hisha;
11 import koma.Hu;
12 import koma.Kaku;
13 import koma.Kin;
14 import koma.Koma;
15
16 public class Ban {
17         private Koma banarray[][] = new Koma[6][6];
18         public Ban() {
19                 set_first_ban();
20         }
21         public Ban(Koma[][] banarray) {
22                 this.banarray = banarray;
23         }
24         public Ban clone() {
25                 Koma array[][] = new Koma[6][6];
26                 for(int i=0;i<6;i++) {
27                         for(int j=0;j<6;j++) {
28                                 array[i][j] = banarray[i][j];
29                         }
30                 }
31                 return new Ban(array);
32         }
33         /**
34          * @return true:同じ, false:違う
35          */
36         public boolean equal(Ban ban) {
37                 for(int i=0;i<6;i++) {
38                         for(int j=0;j<6;j++) {
39                                 if(banarray[i][j]==null) {
40                                         if(ban.get_banarray(i, j)!=null) {
41                                                 return false;
42                                         }
43                                 }else {
44                                         if(!banarray[i][j].equal(ban.get_banarray(i, j))) {
45                                                 return false;
46                                         }
47                                 }
48                         }
49                 }
50                 return true;
51         }
```

```
52          public void save_file(FileWriter file_writer) {
53                  try {
54                          for(int i=0;i<6;i++) {
55                                  for(int j=0;j<6;j++) {
56                                          if(banarray[i][j]==null) {
57                                                  file_writer.write("0\n");
58                                          }else {
59                                                  banarray[i][j].save_file(file_writer);
60                                          }
61                                  }
62                          }
63                  } catch (IOException e) {
64                          // TODO 自動生成された catch ブロック
65                          e.printStackTrace();
66                  }
67          }
68          /**
69           * 正しく読み込めたら true
70           * @return
71           */
72          public boolean load_file(BufferedReader br) {
73                  try {
74                          String str = null;
75                          for(int i=0;i<6;i++) {
76                                  for(int j=0;j<6;j++) {
77                                          str = br.readLine();
78                                          if(str.equals("0")) {
79                                                  banarray[i][j] = null;
80                                          }else {
81                                                  Koma koma = Calc_koma.make_load_Koma(str);
82                                                  if(koma == null) {
83                                                          return false;
84                                                  }
85                                                  banarray[i][j] = koma;
86                                          }
87                                  }
88                          }
89                  } catch (IOException e) {
90                          // TODO 自動生成された catch ブロック
91                          e.printStackTrace();
92                          return false;
93                  }
94                  return true;
95          }
96          /**
97           * 盤面を出力する
98           */
99          public void output_ban() {
100                 System.out.println("盤面を出力します。");
101                 Koma koma = null;
102                 for(int i=1;i<6;i++) {
103                         for(int j=5;j>0;j--) {
104                                 koma = get_banarray(j, i);
105                                 if(koma==null) {
106                                         System.out.print("     ");
107                                 }else {
108                                         koma.output_koma();
```

```
109                                 }
110                         }
111                         System.out.println("");
112                 }
113         }
114         /**
115          * 初期局面を作る。
116          */
117         public void set_first_ban() {
118                 //盤面，全て null にする
119                 for(int i=1;i<6;i++) {
120                         for(int j=1;j<6;j++) {
121                                 set_banarray(i, j, null);
122                         }
123 }
124                 //駒配置
125                 Koma hu1 = new Hu(1);
126                 Koma hu2 = new Hu(2);
127                 //歩
128                 set_banarray(1, 2, hu2);
129                 set_banarray(5, 4, hu1);
130
131         //銀
132         Koma gin1 = new Gin(1);
133         Koma gin2 = new Gin(2);
134         set_banarray(3, 1, gin2);
135         set_banarray(3, 5, gin1);
136         //金
137         Koma kin1 = new Kin(1);
138         Koma kin2 = new Kin(2);
139         set_banarray(2, 1, kin2);
140         set_banarray(4, 5, kin1);
141         //玉
142         Koma gyoku1 = new Gyoku(1);
143         Koma gyoku2 = new Gyoku(2);
144         set_banarray(1, 1, gyoku2);
145         set_banarray(5, 5, gyoku1);
146         //角
147         Koma kaku1 = new Kaku(1);
148         Koma kaku2 = new Kaku(2);
149         set_banarray(4, 1, kaku2);
150         set_banarray(2, 5, kaku1);
151         //飛車
152         Koma hisha1 = new Hisha(1);
153         Koma hisha2 = new Hisha(2);
154         set_banarray(5, 1, hisha2);
155         set_banarray(1, 5, hisha1);
156
157
158         }
159         /**
160          * 盤に駒をセットする。
161          * 駒の place もセットする。
162          * @param a
163          * @param b
164          * @param koma
165          */
```

```
166         public void set_banarray(int a,int b,Koma koma){
167                 if (0<a && a<6 && 0<b && b<6) {
168                         banarray[a][b] = koma;
169                         if(koma!=null) {
170                                 koma.set_place(a,b);
171                         }
172                 }
173         }
174         public Koma get_banarray(int a,int b) {
175                 if (0<a && a<6 && 0<b && b<6) {
176                         return banarray[a][b];
177                 }
178                 return null;
179         }
180
181 }
```

・Board クラス

```
1  package board;
2
3  import java.util.ArrayList;
4  import java.util.Collections;
5
6  import calc_value.Tumi_check;
7  import calc_value.Value;
8  import main.Kihu;
9  import main.Player;
10 import swing.Action_listener_menu;
11
12 /**
13  * ここがメインのクラス。
14  * Board クラスは次の Board クラスを子に持ちどこに行くのがいいか判断する。
15  * player が人間しかいなければ board のこクラスは作らない。
16  * player によって次の手を考えるか、入力を待つか判断する。
17  *
18  */
19
20  public class Board {
21         //手番は局面から取る
22         private Kyokumen kyokumen;
23         private Player player;
24         private Move before_move;//直前の move, 棋譜を保存するときに使う。
25         private ArrayList<Board> board_list;//候補のリスト, 全ての手のリスト
26         private Value value;//評価値,2147480000max
27         private static int max_depth = 2;//何手まで読めるようにするか,1 手読むとは、自分が指して相手が指
   した手の評価をする?。
28         private ArrayList<Kyokumen> kyokumen_list;//千日手チェックに必要, この board の持つ kyokumen は
   入っていない
29         private Kihu kihu;//棋譜を保存する。
30         private static int routine = 0;
31
32         private static long seTime = 0;//処理時間
33         private static int count_tumi = 0;
34
35         public Board(Kyokumen kyokumen,Player player,ArrayList<Kyokumen> kyokumen_list,Kihu kihu) {
36                 this.kyokumen = kyokumen;
37                 this.player = player;
38                 this.kyokumen_list = kyokumen_list;
```

```
39                  board_list = new ArrayList<Board>();
40                  value = new Value();
41                  before_move = null;
42                  this.kihu = kihu;
43          }
44          /**
45           * main not swing よう
46           * @param kyokumen
47           * @param player
48           * @param kyokumen_list
49           */
50          public Board(Kyokumen kyokumen,Player player,ArrayList<Kyokumen> kyokumen_list) {
51                  this.kyokumen = kyokumen;
52                  this.player = player;
53                  this.kyokumen_list = kyokumen_list;
54                  board_list = new ArrayList<Board>();
55                  value = new Value();
56                  before_move = null;
57                  this.kihu = new Kihu(kyokumen);
58          }
59          /**
60           * 局面クラスだけで作る。
61           * その局面が詰みかどうか局面だけから判断したい時に使う
62           * @param kyokumen
63           */
64          public Board(Kyokumen kyokumen) {
65                  this.kyokumen = kyokumen;
66                  this.player = new Player();
67                  this.before_move = null;
68                  this.board_list = null;
69                  this.value = new Value();
70                  this.kyokumen_list = new ArrayList<Kyokumen>();
71                  this.kihu = new Kihu(kyokumen);
72          }
73          /**
74           * 棋譜を読み込む
75           * @param load_file
76           */
77          public void read_kihu(String load_file) {
78                  set_first();
79                  kihu.read_kihu_from_first_kyokumen(load_file);
80                  kyokumen = kihu.get_first_kyokumen().clone();
81          }
82          /**
83           * 局面を読み込む
84           * 局面以外は初期状態にする。
85           */
86          public void read_kyokumen(String load_file) {
87                  set_first();
88                  kyokumen.load_kyokumen(load_file);
89          }
90          /**
91           * 初期画面にする。他のフィールドも初期状態にする。
92           */
93          public void set_first() {
94                  this.kyokumen = new Kyokumen();
95                  this.player = new Player();
```

```
96                  this.before_move = null;
97                  this.board_list = new ArrayList<Board>();
98                  this.value = new Value();
99                  this.kyokumen_list = new ArrayList<Kyokumen>();
100                 this.kihu = new Kihu(kyokumen);
101         }
102         /**
103          * 局面以外はリセット
104          */
105         public void reset_except_kyokumen() {
106                 kyokumen.decide_all_koma_move_place();
107                 this.player = new Player();
108                 this.before_move = null;
109                 this.board_list = new ArrayList<Board>();
110                 this.value = new Value();
111                 this.kyokumen_list = new ArrayList<Kyokumen>();
112                 this.kihu = new Kihu(kyokumen);
113         }
114         /**
115          * 引数の board に変身する
116          * @param board
117          */
118         private void set_board(Board board) {
119                 this.kyokumen = board.get_kyokumen();
120                 //this.player
121                 this.before_move = board.get_before_move();
122                 this.board_list = board.get_board_list();
123                 this.value = board.get_value();
124                 this.kyokumen_list = board.get_kyokumen_list();
125                 this.kihu = board.get_kihu();
126         }
127         public void save_kyokumen(String save_file) {
128                 kyokumen.save_kyokumen(save_file);
129         }
130         public int check_sennitite(ArrayList<Kyokumen> kyokumen_list) {
131                 return Calc_kyokumen_foul.check_sennitite(kyokumen_list, kyokumen);
132         }
133         public int check_sennitite() {
134                 return Calc_kyokumen_foul.check_sennitite(kyokumen_list, kyokumen);
135         }
136         /**
137          *（深さ１で）詰んでいるか確認する
138          * @return 手番の王様が詰んでいる時 true
139          */
140         public boolean check_tumi() {
141                 Board bb = new Board(kyokumen.clone());
142                 Board next_board = bb.decide_next_board_no_parent(1);
143                 if(next_board==null) {
144                         //次に行く場所がない。詰み
145                         return true;
146                 }
147                 return false;
148         }
149
150         /**
151          * 反則ないか判断。反則なら true
152          * @return
```

```
153          */
154         public boolean check_foul() {
155                 return Calc_kyokumen_foul.check_foul_all(kyokumen,before_move);
156         }
157         /**
158          * 全ての駒の move_place を決める
159          */
160         public void decide_all_koma_move_place() {
161                 kyokumen.decide_all_koma_move_place();
162         }
163
164         /**
165          * 局面を出力する。
166          */
167         public void output_kyokumen() {
168                 System.out.println("value: "+value.get_value());
169                 kyokumen.output_kyokumen();
170         }
171
172         /**
173          * Move をもらって戻る。
174          * @param move
175          */
176         public void move_back_kyokumen(Move move) {
177                 kyokumen.move_back_kyokumen(move);
178         }
179         /**
180          * move から次の局面になる board を返す
181          * すでに board_list に作られていればそれを返す
182          * @param move
183          * @return
184          */
185         public Board get_move_next_board(Move move) {
186                 kyokumen.clone().move_next_kyokumen(move);//ここで最終的に move が決まる
187                 if(board_list!=null && !board_list.isEmpty()) {
188                         for(Board b:board_list) {
189                                 if(move.equals_move(b.get_before_move())) {
190                                         return b;
191                                 }
192                         }
193                 }
194                 ArrayList<Kyokumen> kyokumen_list_n = new ArrayList<Kyokumen>();
195                 for(Kyokumen k:kyokumen_list) {
196                         kyokumen_list_n.add(k);
197                 }
198                 //kyokumen_list_n.add(kyokumen.clone());
199                 Board board = new Board(kyokumen.clone(),player,kyokumen_list_n,kihu.clone_kihu());
200                 board.move_next_kyokumen(move);
201                 return board;
202         }
203         /**
204          * move をもらって局面を進める。
205          * board_list にあればその board になる
206          * @param move
207          */
208         public void move_next_kyokumen(Move move) {
209                 kyokumen.move_next_kyokumen(move);//ここで最終的に move が決まる
```

```
210                 if(board_list!=null && !board_list.isEmpty()) {
211                         for(Board b:board_list) {
212                                 if(move.equals_move(b.get_before_move())) {
213                                         set_board(b);
214                                         return;
215                                 }
216                         }
217                 }
218                 kyokumen_list.add(kyokumen.clone());
219                 kihu.add_move(move.clone());
220                 before_move = move;
221         }
222         /**
223          * 次の局面へいく。コンピュータ, 人間共に使う。
224          * @return
225          * 次の board を返す
226          */
227         public Board go_next_board() {
228                 Board board = null;
229                 Move move = null;
230                 if(player.get_player(get_teban())) {
231                         //コンピュータ, 候補手の数だけ探索する。
232                         board = decide_next_board_no_parent(1);
233                 }else {
234                         //人間,
235                         //正しい move を受け取る。反則は関係ない。
236                         Calc_move_human calc_move_human = new Calc_move_human(kyokumen);
237                         move = calc_move_human.get_move();
238                         kyokumen_list.add(kyokumen.clone());
239                         board = new Board(kyokumen, player,kyokumen_list);
240                         board.move_next_kyokumen(move);
241                         if(board != null) {
242                                 board.set_before_move(move);//いらない///
243                         }
244                 }
245                 return board;
246         }
247         /**
248          * 次の局面がなければ 0 返す
249          * @return
250          */
251         public int go_next_board_from_swing() {
252                 System.out.println("calc next te!");///
253                 //Board board = decide_next_board_no_parent(1);
254                 Board board = select_routine();
255                 //System.out.println("使えてます");
256                 if(board==null) {
257                         return 0;
258                 }
259                 set_board(board);
260                 return 1;
261         }
262
263         /*
264          *先手詰将棋ルーチン
265          */
266
```

```
267
268         public Board sente_tumi_go_next() {
269                 Board board = null;
270                 Move move = Tumi_check.check_tumi_n_te(kyokumen,3);
271                 long startTime = 0;
272                 long endTime = 0;
273                 if(get_teban()==1) {
274                         //先手
275                         startTime = System.currentTimeMillis();
276                 if(move==null) {
277                         //積んでいなければ
278                         board = decide_next_board_no_parent(1);
279                         //System.out.println("使えてます");
280                         count_tumi++;
281                         System.out.println("先手" + count_tumi + "回目");
282                         endTime = System.currentTimeMillis();
283                         }else {
284                         //積んでいたら
285                         board = new Board(kyokumen, player,kyokumen_list);
286                         board.move_next_kyokumen(move);
287                         System.out.println("積んでます");
288                         count_tumi++;
289                         System.out.println("先手" + count_tumi + "回目");
290                         endTime = System.currentTimeMillis();
291                         //System.out.println("tumi_routine");
292                         }
293
294                 }else {
295                         //後手
296                         board = decide_next_board_no_parent(1);
297                 }
298                 long time = endTime - startTime;
299                 seTime += time;
300                 return board;
301         }
302
303
304         /*
305          * 後手詰め手ルーチン
306          */
307
308
309         public Board gote_tumi_go_next() {
310                 Board board = null;
311                 Move move = Tumi_check.check_tumi_n_te(kyokumen,5);
312                 long startTime = 0;
313                 long endTime = 0;
314                 if(get_teban()==1) {
315                         //先手
316                         board = decide_next_board_no_parent(1);
317                 }else {
318                         //後手
319                         startTime = System.currentTimeMillis();
320                 if(move==null) {
321                         //積んでいなければ
322                         board = decide_next_board_no_parent(1);
323                         //System.out.println("使えてます");
```

```
324                         count_tumi++;
325                         System.out.println("後手" + count_tumi + "回目");
326                         endTime = System.currentTimeMillis();
327                         }else {
328                         //積んでいたら
329                         board = new Board(kyokumen, player,kyokumen_list);
330                         board.move_next_kyokumen(move);
331                         count_tumi++;
332                         System.out.println("後手" + count_tumi + "回目");
333                         endTime = System.currentTimeMillis();
334                         //System.out.println("tumi_routine");
335                         }
336                 }
337                 long time = endTime - startTime;
338                 seTime += time;
339                 return board;
340         }
341
342
343         /*
344          * 両手詰め手ルーチン
345          */
346
347         public Board ryote_tumi_go_next() {
348                 Board board = null;
349                 Move move = Tumi_check.check_tumi_n_te(kyokumen,3);
350                 long startTime = 0;
351                 long endTime = 0;
352                 if(get_teban()==1) {
353                         //先手
354                         startTime = System.currentTimeMillis();
355                         if(move==null) {
356                                 //積んでいなければ
357                                 board = decide_next_board_no_parent(1);
358                                 //System.out.println("使えてます");
359                                 count_tumi++;
360                                 System.out.println("先手" + count_tumi + "回目");
361                                 endTime = System.currentTimeMillis();
362                                 }else {
363                                 //積んでいたら
364                                 board = new Board(kyokumen, player,kyokumen_list);
365                                 board.move_next_kyokumen(move);
366                                 count_tumi++;
367                                 System.out.println("先手" + count_tumi + "回目");
368                                 endTime = System.currentTimeMillis();
369                                 //System.out.println("tumi_routine");
370                                 }
371
372                 }else {
373                         //後手
374                         startTime = System.currentTimeMillis();
375                 if(move==null) {
376                         //積んでいなければ
377                         board = decide_next_board_no_parent(1);
378                         //System.out.println("使えてます");
379                         count_tumi++;
380                         System.out.println("後手" + count_tumi + "回目");
```

```
                        endTime = System.currentTimeMillis();
                        }else {
                        //積んでいたら
                        board = new Board(kyokumen, player,kyokumen_list);
                        board.move_next_kyokumen(move);
                        count_tumi++;
                        System.out.println("後手" + count_tumi + "回目");
                        endTime = System.currentTimeMillis();
                        //System.out.println("tumi_routine");
                        }
                }
                long time = endTime - startTime;
                seTime += time;
                return board;
        }



        /*
         * 両手αβ
         */

        public Board ab_go_next() {
                Board board = null;
                long startTime = 0;
                long endTime = 0;
                if(get_teban()==1) {
                        //先手
                        startTime = System.currentTimeMillis();
                        board = decide_next_board_no_parent(1);
                        count_tumi++;
                        System.out.println("先手" + count_tumi + "回目");
                        endTime = System.currentTimeMillis();


                }else {
                        //後手
                        startTime = System.currentTimeMillis();
                        board = decide_next_board_no_parent(1);
                        count_tumi++;
                        System.out.println("先手" + count_tumi + "回目");
                        endTime = System.currentTimeMillis();


                }
                long time = endTime - startTime;
                seTime += time;
                return board;
        }



        public static long get_seTime() {
                return seTime;
        }

        public static int get_count() {
```

```
438                 return count_tumi;
439         }
440
441         public static void reset_count_time() {
442                 seTime = 0;
443                 count_tumi = 0;
444         }
445
446
447
448         /*
449          * ルーチンを選べる
450          * 1ならαβ法
451          * 2なら詰将棋ルーチン
452          *
453          */
454         public Board select_routine() {
455                 Board board = null;
456                 if(routine == 1) {
457                         //System.out.println("αβ法使用中");
458                         board = ab_go_next();
459                         return board;
460                 }else if(routine == 2){
461                         //System.out.println("先手詰みルーチン使用中");
462                         board = sente_tumi_go_next();
463                         return board;
464                 }else if(routine == 3){
465                         //System.out.println("後手詰みルーチン使用中");
466                         board = gote_tumi_go_next();
467                         return board;
468                 }else {
469                         //System.out.println("両手詰みルーチン使用中");
470                         board = ryote_tumi_go_next();
471                         return board;
472                 }
473         }
474         /**
475          * board_list を先手なら評価値の高い順に並び替える（後手なら低い順）
476          */
477         private void sort_board_list() {
478                 if(get_teban()==1) {
479                         Collections.sort(board_list, new ValueComparator_gote());
480                 }else {
481                         Collections.sort(board_list,new ValueComparator_sente());
482                 }
483         }
484
485
486         /**
487          * board_list 作る。選ぶ。
488          * board_list から次の局面を選んでその局面を返す。
489          * a-b 法
490          * depth は 1 から。1 なら 1 手読む
491          * value の値も更新する。
492          * @return 次に行く局面を返す
493          */
494         public Board decide_next_board_no_parent(int depth) {
```

```
495                     //board_list は一回しか作らない
496                     if(board_list==null || board_list.isEmpty()) {
497                             make_board_list();
498                     }
499                     select_board_list();
500                     if(board_list==null || board_list.isEmpty()) {
501                             //負けている
502                             value.set_determe(true);
503                             if(get_teban()==1) {
504                                     value.set_value(-100);
505                             }else {
506                                     value.set_value(100);
507                             }
508                             return null;
509                     }
510                     sort_board_list();
511
512                             if(depth==1) {
513                                     System.out.println("sorted board list size: "+board_list.size());
514                             }
515
516                     if(depth>=max_depth) {
517                             Board board = board_list.get(0);//1 手先で最善のやつ
518                             //ここで value の値も一つ下の値と同じに更新する。
519                             value.set_value_all(board.get_value());
520                             return board;
521                     }
522                     //a-b 法,board_list の中で一番いいのを選びたい。
523                     Board next_board = null;//次の局面でその先の選択肢があるか？
524                     Board board_expect = null;//次になりそうな board
525                      ArrayList<Board> board_list_expect = new ArrayList<Board>();
526                      ArrayList<Board> board_list_expect2 = new ArrayList<Board>();
527                     int n = 1;///
528                     for(Board board:board_list) {
529                             if(depth==1) {
530                                     System.out.println("count "+n++);///
531                             }
532                             if(board_expect == null) {
533                                     //初めての時
534                                     if(!board.get_value().get_determe()) {
535                                             //board の評価値が決まっていない時はその下を探索する。評価値が決
    定されている時は下を探索する必要はない
536                                             next_board = board.decide_next_board_no_parent(depth+1);
537                                             if(next_board==null) {
538                                                     //勝っている.board を選べば次相手は負ける。board は負け
    ている
539                                                     value.set_win(get_teban());
540                                                     return board;
541                                             }
542                                     }
543                                     ///勝敗決まっているときはその先を計算する必要ないかもしれないので，省略し
    た方が良い??
544                                     board_expect = board;
545                                     board_list_expect.add(board_expect);
546                             }else {
547                                     //board_expect がある時
548                                     if(!board.get_value().get_determe()) {
```

```
549                                     //board の評価値が決まっていない時はその下を探索する。評価値が決
    定されている時は下を探索する必要はない
550                                     next_board = board.decide_next_board_have_parent(depth+1, board_expect);
551                                     if(next_board==null) {
552                                             //勝っている.board を選べば次相手は負ける。board は負け
    ている
553                                             value.set_win(get_teban());
554                                             return board;
555                                     }
556                             }
557                             if(compare_board(board, board_expect)) {
558                                     //board の方がいい時
559                                     board_expect = board;
560                                     board_list_expect.add(board_expect);
561                             }
562                     }
563             }
564             value.set_value_all(board_expect.get_value());
565             //System.out.println(board_list_expect.size());
566             if((board_list_expect.size()<=1)){
567                     return board_expect;
568             }else {
569
570                      board_list_expect2.add(board_list_expect.get(board_list_expect.size()-1));
571                      board_list_expect2.add(board_list_expect.get(board_list_expect.size()-2));
572             }
573
574             Collections.shuffle(board_list_expect2);
575             board_expect = board_list_expect2.get(0);
576             return board_expect;
577     }
578     /**
579      * board_list 作る。選ぶ。
580      * board_list から次の局面を選んでその局面を返す。
581      * a-b 法
582      * depth は 1 から。1 なら 1 手読む
583      * value の値も更新する。
584      * 自分の仮の評価値と親の評価値を比較して、ここで自分の評価値の方がよければ終了!
585      * @return そんなに意味はない?
586      */
587     private Board decide_next_board_have_parent(int depth,Board board_parent) {
588             if(board_list==null || board_list.isEmpty()) {
589                     make_board_list();
590             }
591             select_board_list();
592             if(board_list==null || board_list.isEmpty()) {
593                     //負けている
594                     value.set_determe(true);
595                     if(get_teban()==1) {
596                             value.set_value(-100);
597                     }else {
598                             value.set_value(100);
599                     }
600                     return null;
601             }
602             sort_board_list();
603             if(depth>=max_depth) {
```

```
604                             Board board = board_list.get(0);//1 手先で最善のやつ
605                             //ここで value の値も一つ下の値と同じに更新する。
606                             value.set_value_all(board.get_value());
607                             return board;
608                     }
609                     //a-b 法,board_list の中で一番いいのを選びたい。親の評価値より自分の評価値がよければ終了!
610                     Board next_board = null;
611                     Board board_expect = null;//次になりそうな board
612                     for(Board board:board_list) {
613                             if(board_expect == null) {
614                                     //初めての時
615                                     if(!board.get_value().get_determe()) {
616                                             //board の評価値が決まっていない時はその下を探索する。評価値が決
    定されている時は下を探索する必要はない
617                                             next_board = board.decide_next_board_no_parent(depth+1);
618                                             if(next_board==null) {
619                                                     //勝っている.board を選べば次相手は負ける。board は負け
    ている
620                                                     value.set_win(get_teban());
621                                                     return board;
622                                             }
623                                     }
624                                     ///勝敗決まっているときはその先を計算する必要ないかもしれないので, 省略し
    た方が良い??
625                                     //親の評価値より自分の評価値がよければ終了!
626                                     if(compare_board(board, board_parent)) {
627                                             value.set_value_all(board.get_value());
628                                             return board;
629                                     }
630                                     board_expect = board;
631                             }else {
632                                     //board_expect がある時
633                                     if(!board.get_value().get_determe()) {
634                                             //board の評価値が決まっていない時はその下を探索する。評価値が決
    定されている時は下を探索する必要はない
635                                             next_board = board.decide_next_board_have_parent(depth+1, board_expect);
636                                             if(next_board==null) {
637                                                     //勝っている.board を選べば次相手は負ける。board は負け
    ている
638                                                     value.set_win(get_teban());
639                                                     return board;
640                                             }
641                                     }
642                                     if(compare_board(board, board_expect)) {
643                                             //board の方がいい時
644                                             //親の評価値より自分の評価値がよければ終了!
645                                             if(compare_board(board, board_parent)) {
646                                                     value.set_value_all(board.get_value());
647                                                     return board;
648                                             }
649                                             board_expect = board;
650                                     }
651                             }
652                     }
653                     value.set_value_all(board_expect.get_value());
654                     return board_expect;
655             }
```

```
656         /**
657          *
658          * @param b1
659          * @param b2
660          * @return b1 がよければ true,(先手なら大きい、後手なら小さい)
661          */
662         private boolean compare_board(Board b1,Board b2) {
663                 if(get_teban()==1) {
664                         return ValueComparator_sente.cmpare_board_sente(b1, b2);
665                 }else if(get_teban()==2) {
666                         return ValueComparator_gote.compare_board_gote(b1, b2);
667                 }
668                 System.out.println("error:teban in Board class compare_board method");
669                 return true;
670         }
671         /**
672          * 勝っているのがあればそれ一つにする
673          * 負けているのは除外する
674          * 並び替える前に呼ばれる。
675          */
676         private void select_board_list() {
677                 ArrayList<Board> remove_list = new ArrayList<Board>();
678                 Value value = null;
679                 for(Board board:board_list) {
680                         value = board.get_value();
681                         if(value.get_determe() && value.get_value()!=0) {
682                                 //勝敗が決まっている。
683                                 //引き分けは除かない
684                                 if(value.get_win(get_teban())) {
685                                         //勝ち
686                                         ArrayList<Board> win_list = new ArrayList<Board>();
687                                         win_list.add(board);
688                                         board_list = win_list;
689                                         return;
690                                 }else {
691                                         //負け
692                                         remove_list.add(board);
693                                 }
694                         }
695                 }
696                 for(Board board:remove_list) {
697                         board_list.remove(board);
698                 }
699         }
700         /**
701          * 全ての手のリストを作る
702          */
703         private void make_board_list() {
704                 board_list = Make_board_list.get_next_board_list(this);
705                 //それぞれの board クラスでその局面での評価値決める
706                 for(Board board:board_list) {
707                         board.set_value_this();
708                 }
709         }
710         public void set_value_this() {
711                 value.calc_value_present(kyokumen,kyokumen_list,before_move);
712         }
```

```
713         public void output_value_test() {
714                 value.calc_value_present(kyokumen,kyokumen_list,before_move);
715                 System.out.println("評価値は:"+value.get_value());
716         }
717         public String get_teban_s() {
718                 if(get_teban()==1) {
719                         return "先手";
720                 }else if(get_teban()==2){
721                         return "後手";
722                 }
723                 return "error";
724         }
725         public String get_teban_opposite_s() {
726                 if(get_teban()==2) {
727                         return "先手";
728                 }else if(get_teban()==1){
729                         return "後手";
730                 }
731                 return "error";
732         }
733         public Move get_before_move() {
734                 return before_move;
735         }
736         public Value get_value() {
737                 return value;
738         }
739         public void set_before_move(Move move) {
740                 this.before_move = move;
741         }
742         public Kyokumen get_kyokumen() {
743                 return kyokumen;
744         }
745         public ArrayList<Board> get_board_list(){
746                 return board_list;
747         }
748         public int get_teban() {
749                 return kyokumen.get_teban();
750         }
751         public void set_player_sente(boolean sente) {
752                 this.player.set_player_sente(sente);
753         }
754         public void set_player_gote(boolean gote) {
755                 this.player.set_player_gote(gote);
756         }
757         public String get_player_both() {
758                 return this.player.get_player_both();
759         }
760         public boolean get_teban_player() {
761                 if(get_teban()==1) {
762                         return this.player.get_player_sente();
763                 }
764                 return this.player.get_player_gote();
765         }
766         public ArrayList<Kyokumen> get_kyokumen_list(){
767                 return kyokumen_list;
768         }
769         public Kihu get_kihu() {
```

```
770                 return kihu;
771         }
772         public void save_kihu(String file_name) {
773                 kihu.save_kihu_from_first_kyokumen(file_name);
774         }
775         public void delete_last_kyokumen_list() {
776                 kyokumen_list.remove(kyokumen_list.size()-1);
777         }
778         /**
779          * cp が含まれていれば true
780          * @return
781          */
782         public boolean contain_cp() {
783                 return player.contain_cp();
784         }
785         public void set_kyokumen(Kyokumen kyokumen) {
786                 this.kyokumen = kyokumen;
787         }
788         public void set_kihu(Kihu kihu) {
789                 this.kihu = kihu;
790         }
791
792         public static void set_depth_max(){
793                 Board.max_depth = Action_listener_menu.get_dep2();
794         }
795
796         public static void set_routine() {
797                 Board.routine = Action_listener_menu.get_routine();
798         }
799
800  }
```

・Calc_kyokumen_foul クラス

```
1  package board;
2
3  import java.util.ArrayList;
4
5  import calc_value.Tumi_check;
6  import koma.Koma;
7  import main.Calc;
8
9  /**
10  * kyokumen に反則がないかを調べるクラス。千日手も調べる
11  * 本来は kyokumen クラスに書きたいが、長く成るのでこのクラスを作った
12  *
13  */
14
15  public class Calc_kyokumen_foul {
16         /**
17          * 千日手がかどうかを調べる
18          * kyokumen_list に kyokumen は入っている?
19          * @return 0:千日手ではない,1:千日手,2; 連続王手の千日手手番が王手している,3:連続王手の千日手手番が
    王手されている
20          */
21         public static int check_sennitite(ArrayList<Kyokumen> kyokumen_list,Kyokumen kyokumen) {
22                 //kyokumen_list に同じ局面があるか調べる。
23                 if(kyokumen_list==null || kyokumen_list.isEmpty()) {
24                         return 0;
```

```
25                  }
26                  int same_kyokumen_count = 0;
27                  for(Kyokumen k:kyokumen_list) {
28                          if(k.equal(kyokumen)) {
29                                  same_kyokumen_count++;
30                          }
31                  }
32                  if(same_kyokumen_count<4) {
33                          return 0;
34                  }
35                  for(Kyokumen k:kyokumen_list) {
36                          k.output_kyokumen();
37                  }
38                  //連続王手が無いか調べる，手番の玉に王手がかかっているか
39                  if(check_oute(kyokumen, kyokumen.get_teban())) {
40                          //手番の玉に王手がかかっている。2 手ずつ戻って王手が連続しているか調べる。
41                          int number = kyokumen_list.size()-2;
42                          Kyokumen kyokumen_s = null;
43                          while(true) {
44                                  kyokumen_s = kyokumen_list.get(number);
45                                  if(kyokumen_s.equal(kyokumen)){
46                                          //連続王手のまま一つ前の同じ局面まで来ている
47                                          return 3;
48                                  }
49                                  if(!check_oute(kyokumen_s, kyokumen.get_teban())) {
50                                          //王手がかかっていない局面があれば連続王手では無い
51                                          break;
52                                  }
53                                  number = number - 2;
54                                  if(number < 0) {
55  System.out.println("error: check_sennitite sennitite but no same kyokumen");
56                                          return 0;
57                                  }
58                          }
59                  }
60                  //連続王手、手番が王手している。
61                  int number = kyokumen_list.size()-1;//一つ前の局面
62                  Kyokumen kyokumen_first = kyokumen_list.get(number);
63                  if(check_oute(kyokumen_first, kyokumen_first.get_teban())) {
64                          Kyokumen kyokumen_s = null;
65                          number = number - 2;
66                          while(true) {
67                                  kyokumen_s = kyokumen_list.get(number);
68                                  if(kyokumen_s.equal(kyokumen_first)){
69                                          //連続王手のまま一つ前の同じ局面まで来ている
70                                          return 2;
71                                  }
72                                  if(!check_oute(kyokumen_s, kyokumen_s.get_teban())) {
73                                          //一つ前の局面で相手玉に王手がかかっていない。
74                                          break;
75                                  }
76                                  number = number - 2;
77                                  if(number < 0) {
78  System.out.println("error: check_sennitite sennitite but no same kyokumen");
79                                          return 0;
80                                  }
81                          }
```

```
82                  }
83                  return 1;
84          }
85          /**
86           * 今の局面に反則がないか確認する
87           * 反則がなければ false, 反則があれば (局面としておかしければ)true 返す
88           */
89          public static boolean check_foul_all(Kyokumen kyokumen,Move before_move) {
90                  if(check_koma_number(kyokumen)) {
91                          //System.out.println(kyokumen.get_koma_array());
92                          System.out.println("foul koma number!!");
93                          return true;
94                  }
95                  if(check_dontmove(kyokumen)) {
96                          //System.out.println("foul!! there are koma that dont move!!");
97                          return true;
98                  }
99                  if(check_nihu(kyokumen)) {
100                         //System.out.println("there are nihu!!");
101                         return true;
102                 }
103                 //王手放置は反則
104                 if(check_oute_ignore(kyokumen)) {
105
106                         //System.out.println("oute ignore!!");
107                         return true;
108                 }
109                 //打ち歩詰め
110                 if(before_move!=null) {
111                         if(check_utihuzume(kyokumen, before_move)) {
112                                 //System.out.println("utihuzume!!");
113                                 return true;
114                         }
115                 }
116                 return false;
117
118         }
119         /**
120          * その局面が打ち歩詰めになっているか。打ち歩詰めなら反則。
121          * @param kyokumen
122          * @param before_move
123          * @return 打ち歩詰めなら true, 普通は false
124          */
125         public static boolean check_utihuzume(Kyokumen kyokumen,Move before_move) {
126                 //最後に歩を打っているか確認する
127                 if(before_move.get_before_place_b()==1) {
128                         if(before_move.get_before_place_a()==10 || before_move.get_before_place_a()==20) {
129                                 //最後に歩を打っている。その局面で手番の玉が詰んでいるか調べる
130                                 ///詰みチェックをしなくてはいけない///
131                                 if(Tumi_check.check_tumi_teban(kyokumen)) {
132                                         return true;
133                                 }
134                         }
135                 }
136                 return false;
137         }
138         /**
```

```
139          * 手番に王手がかかっているか確認し、王手がかかって入れば true
140          * 王手がなければ false
141          * @return
142          */
143         public static boolean check_oute(Kyokumen kyokumen,int teban) {
144                 int place_gyoku = kyokumen.get_place_gyoku(teban);
145                 int teban_r = Calc.change_teban(teban);
146                 ArrayList<Koma> my_koma_list = kyokumen.get_teban_koma_list_all(teban_r);
147                 //System.out.println(my_koma_list);
148                 for(Koma koma:my_koma_list) {
149                         for(int place:koma.get_move_place()) {
150                                 //System.out.println(koma);
151                                 //System.out.println(place);
152                                 if(place==place_gyoku) {
153                                         //System.out.println(koma);
154                                         //System.out.println(place);
155                                         //System.out.println(place_gyoku);
156
157                                         return true;
158                                 }
159                         }
160                 }
161                 return false;
162         }
163         /**
164          * 王手放置は反則負け
165          * 王手放置をして入れば true
166          * 相手の玉に王手がかかって入れば true
167          * @return
168          */
169         private static boolean check_oute_ignore(Kyokumen kyokumen) {
170                 int teban_r = Calc.change_teban(kyokumen.get_teban());
171                 return check_oute(kyokumen,teban_r);
172         }
173         /**
174          * コマの数を数える。
175          * 数があっていれば false, おかしければ true
176          * コマ落ちも使うならあってはいけないメソッド。コマの枚数が変わることはありえないので正しく動くようになればいらない
177          */
178
179         private static boolean check_koma_number(Kyokumen kyokumen) {
180                 int i,j;
181                 int[] count_koma = new int[7];
182                 for(i=0;i<7;i++) {
183                         count_koma[i] = 0;//初期化
184                 }
185                 for(i=0;i<7;i++) {
186                         //System.out.println(count_koma[i]);
187                         count_koma[i] += kyokumen.get_sente_komadai().get_koma_number(i);
188                         //System.out.println("sente"+count_koma[i]);
189                         count_koma[i] += kyokumen.get_gote_komadai().get_koma_number(i);//駒台の駒数える
190                         //System.out.println("gote"+count_koma[i]);
191                 }
192                 Koma koma = null;
193                 int n = 0;
```

```
194                 for(i=1;i<6;i++) {
195                         for(j=1;j<6;j++) {
196                                 koma = kyokumen.get_koma_from_place(i, j);
197                                 if(koma!=null) {
198                                         n = koma.get_koma_number() % 10;
199                                         count_koma[n] += 1;
200                                         //System.out.println(count_koma[n]);
201                                 }
202                         }
203                 }
204                 if(count_koma[1] != 2) {
205                         System.out.println("check_foul error count hu");
206                         //System.out.println(count_koma[1]);
207                         return true;
208                 }else if(count_koma[2] != 2) {
209                         System.out.println("check_foul error count gin");
210                         //System.out.println(count_koma[2]);
211                         return true;
212                 }else if(count_koma[3] != 2) {
213                         System.out.println("check_foul error count kin");
214                         //System.out.println(count_koma[3]);
215                         return true;
216                 }else if(count_koma[4] != 2) {
217                         System.out.println("check_foul error count gyoku");
218                         //System.out.println(count_koma[4]);
219                         return true;
220                 }else if(count_koma[5] != 2) {
221                         System.out.println("check_foul error count kaku");
222                         //System.out.println(count_koma[5]);
223                         return true;
224                 }else if(count_koma[6] != 2) {
225                         System.out.println("check_foul error count hisha");
226                         //System.out.println(count_koma[6]);
227                         return true;
228                 }
229                 return false;
230         }
231
232
233         /**
234          * 動かせない駒（歩、香、桂）があるかチェック
235          * なければ false, あれば true
236          */
237         private static boolean check_dontmove(Kyokumen kyokumen) {
238                 for(Koma koma:kyokumen.get_koma_array()) {
239                         if(koma.able_to_move()) {
240                                 return true;//一つでも反則あればダメ。
241                         }
242                 }
243                 return false;
244         }
245
246         /**
247          * 二歩チェック
248          * 反則なければ false, 二歩あれば true
249          */
250         private static boolean check_nihu(Kyokumen kyokumen) {
```

```
251                 //二歩チェック
252                 int[] sente = {0,0,0,0,0,0};
253                 int[] gote = {0,0,0,0,0,0};
254                 for(Koma koma:kyokumen.get_koma_array()) {
255                         if(koma.get_koma_number()==1) {
256                                 int n = koma.get_place_a();
257                                 if(koma.get_teban()==1) {
258                                         sente[n] += 1;
259                                 }else {
260                                         gote[n] += 1;
261                                 }
262                         }
263                 }
264                 for(int i=0;i<6;i++) {
265                         if(sente[i]>1 || gote[i]>1) {
266                                 return true;//二歩
267                         }
268                 }
269                 return false;
270         }
271  }
```

・Calc_move_human クラス

```
1  package board;
2
3  import koma.Koma;
4
5  /**
6   * 人間の入力ができるようにする。
7   *
8   */
9  public class Calc_move_human {
10         private Kyokumen kyokumen;
11         public Calc_move_human(Kyokumen kyokumen) {
12                 this.kyokumen = kyokumen;
13         }
14         /**
15          * 正しい入力の move を返す
16          * @return
17          */
18         public Move get_move() {
19                 //move を作る。
20                 Move move = null;
21                 while(true) {
22                         move = make_move();
23                         //move が正しいか判定。正しくなければやり直し。正しければそれを return
24                         if(check_go_next_kyokumen(move)) {
25                                 //System.out.println("next kyokumen");
26                                 break;
27                         }
28                         System.out.println("入力が正しくないです。もう一度やり直してください。");//76
29                 }
30                 //成るかどうかの確認
31                 Koma koma = kyokumen.get_koma_from_place(move.get_before_place_a(), move.get_before_place_b());
32                 if(move.check_naru(koma)) {
33                         move.set_naru(true);
34                 }
35                 return move;
```

```
36          }
37          /**
38           * 次の局面にいけるか確認する
39           * 入力は Move クラス。見るのは before_place と after_place だけ。
40           * いけたら true, いけなければ false
41           * @return
42           */
43          private boolean check_go_next_kyokumen(Move move) {
44                  //before が正しいかチェック
45                  int before_a = move.get_before_place_a();
46                  int before_b = move.get_before_place_b();
47                  if(!check_go_before_place(before_a, before_b)) {
48                          return false;
49                  }
50                  //after_place が正しいか確認する。Koma クラスの move_place を見れば良い
51                  Koma koma = kyokumen.get_koma_from_place(before_a, before_b);
52                  int a = move.get_after_place_a();
53                  int b = move.get_after_place_b();
54                  //持ち駒の時は盤が null かだけ見れば良い。
55                  if(koma.get_motigoma()) {
56                          //a,b が 0 はだめ
57                          if(a==0 || b==0) {
58                                  return false;
59                          }
60                          Koma after_koma = kyokumen.get_banarray(a,b);
61                          if(after_koma==null) {
62                                  return true;
63                          }
64                          return false;
65                  }
66                  if(koma.contain_move_place(a*10+b)) {
67                          return true;
68                  }
69                  System.out.println("after の入力が違います。check_go_next_kyokumen");
70                  return false;
71
72          }
73          /**
74           * check_go_next_kyokumen メソッドで使う。before_a,before_b の場所に駒があるか確認する。
75           * @return
76           */
77          private boolean check_go_before_place(int before_a,int before_b) {
78                  if(before_a > 5) {
79                          //駒台のとき
80                          if((before_a/10) == get_teban()) {
81                                  //手番の駒台を指している。駒台に駒があるか確認する。
82                                  if(kyokumen.get_koma_from_place(before_a, before_b) != null) {
83                                          //駒がある
84                                          return true;
85                                  }else {
86                                          //駒がない
87                                          System.out.println("error: beforepalce is not true" +
88                                          "in Calc_move class check_go_before_place method");
89                                          return false;
90                                  }
91                          }else {
92                                  //手番でない駒台を指しているので入力がおかしい。
```

```
93                                  System.out.println("before の入力が違います。Calc_move check_go_before_place");
94                                  return false;
95                          }
96                  }else {
97                          //盤面の駒を動かすとき
98                          Koma koma = kyokumen.get_banarray(before_a, before_b);
99                          if(koma == null) {
100                         System.out.println("before の入力が違います。ban koma==nullCalc_move"+
101                         "check_go_before_place"+before_a+before_b);
102                                 return false;
103                         }
104                         if(koma.get_teban()==get_teban()) {
105                                 //手番の駒がある
106                                 return true;
107                         }
108                         System.out.println("before の入力が違います。teban Calc_move "
109                         +"check_go_before_place"+before_a+before_b);
110                         return false;
111                 }
112         }
113         /**
114          * move クラスを作る。
115          * コンピュータか人間かによって作り方違う。
116          * 駒を動かせる move を返す。人間の入力の時は二歩などの反則はあっていい。
117          * @return
118          */
119         private Move make_move() {
120                 if(get_teban()==1) {
121                         System.out.println("先手番です。");
122                 }else if(get_teban()==2){
123                         System.out.println("後手番です。");
124                 }
125                 Move move = new Move();
126                 move.input_move();
127                 //成るかどうか確認していない。入力は間違っていてもいい
128                 return move;
129         }
130         private int get_teban() {
131                 return kyokumen.get_teban();
132         }
133 }
```

・Free_kyokumen クラス

```
1  package board;
2
3  import koma.Calc_koma;
4  import koma.Koma;
5
6  /**
7   * 局面を自由に編集できるようにするクラス。
8   *
9   */
10 public class Free_kyokumen extends Kyokumen{
11         public Free_kyokumen(Kyokumen kyokumen) {
12                 set_ban_komadai(kyokumen.get_ban().clone(), kyokumen.get_sente_komadai().clone(),
13                 kyokumen.get_gote_komadai().clone());
14         }
15         public Kyokumen get_kyokumen(int teban) {
```

```
16                  Kyokumen kyokumen = new Kyokumen(get_ban(),get_sente_komadai(),get_gote_komadai(),teban);
17                  kyokumen.set_koma_array();
18                  kyokumen.decide_all_koma_move_place();
19                  return kyokumen;
20          }
21          /**
22           * 駒を動かす
23           * @param before_place
24           * @param after_place
25           * 駒台に置く時は取った駒返す
26           */
27          public Koma move(int before_place,int after_place) {
28                  //before_place の駒はなくす。駒台なら減らす
29                  Koma koma = get_koma_from_place(before_place);
30                  if(koma == null) {
31                          System.out.println("error: free kyokumen move: "+before_place);
32                          return null;
33                  }
34                  if(before_place/10 < 10) {
35                          //動かす前は盤上の駒
36                          set_koma_from_place(null, before_place/10, before_place%10);
37                  }else {
38                          decrease_motigoma(koma);//前の場所を無くした。
39                  }
40                  //同じ場所なら成る反転させる。
41                  if(before_place==after_place) {
42                          if(koma.able_to_naru() || koma.get_koma_number()>10) {
43                                  int koma_number = koma.get_koma_number();
44                                  if(koma_number<10) {
45                                          koma_number += 10;
46                                  }else {
47                                          koma_number -= 10;
48                                  }
49                                  Koma koma_n = Calc_koma.make_koma(koma_number, koma.get_teban());
50                                  set_koma_from_place(koma_n, after_place/10, after_place%10);
51                          }else {
52                                  //成れない駒の時はなにもしない
53                                  set_koma_from_place(koma, after_place/10, after_place%10);
54                          }
55                          return null;
56                  }
57                  set_koma_from_place(koma, after_place/10, after_place%10);
58                  if(after_place/10 < 10) {
59                          //盤上に移動
60                          return null;
61                  }
62                  return koma;
63          }
64
65  }
```

・Komadai クラス

```
1   package board;
2
3   import java.io.BufferedReader;
4   import java.io.FileWriter;
5   import java.io.IOException;
6   import java.util.ArrayList;
```

```
7
8   import koma.Calc_koma;
9   import koma.Koma;
10  import main.Calc;
11
12  /**
13   * 駒台を保存するクラス。
14   *
15   */
16  public class Komadai {
17          //0:なし 1:歩 2:銀 3:金 4:玉 5:角 6:飛車
18          private int komadai[] = new int[7];
19          private int teban;
20          //private ArrayList<Koma> koma_array;//Koma クラスのポインタを入れておく。一旦使わない
21          public Komadai(int[] komadai,int teban) {
22                  this.komadai = komadai;
23                  this.teban = teban;
24          }
25          public Komadai(int teban) {
26                  for(int i=0;i<7;i++) {
27                          //最初は駒台に駒はない
28                          komadai[i] = 0;
29                  }
30                  this.teban = teban;
31          }
32          public Komadai clone() {
33                  int dai[] = new int[7];
34                  for(int i=0;i<7;i++) {
35                          dai[i] = komadai[i];
36                  }
37                  return new Komadai(dai,teban);
38          }
39          /**
40           * @param komadai 比較する駒台
41           * @return true:同じ,false:違う
42           */
43          public boolean equal(Komadai komadai_n) {
44                  if(this.teban != komadai_n.get_teban()) {
45                          return false;
46                  }
47                  for(int i=0;i<7;i++) {
48                          if(komadai[i] != komadai_n.get_koma_number(i)) {
49                                  return false;
50                          }
51                  }
52                  return true;
53          }
54          /**
55           * ファイルに保存する
56           * @param file_writer
57           */
58          public void save_file(FileWriter file_writer) {
59                  try {
60                          file_writer.write(get_teban_s()+"\n");
61                          for(int i=0;i<7;i++) {
62                                  file_writer.write(komadai[i] + ",");
63                          }
```

```
64                      file_writer.write("\n");
65              } catch (IOException e) {
66                      // TODO 自動生成された catch ブロック
67                      e.printStackTrace();
68              }
69      }
70      /**
71       * 正しく読み込めたら true
72       * @return
73       */
74      public boolean load_file(BufferedReader br) {
75              try {
76                      //手番があっているか確認
77                      String str = br.readLine();
78                      if(Calc.isNumber(str)) {
79                              int teban = Integer.parseInt(str);
80                              if(this.teban != teban) {
81                                      return false;
82                              }
83                      }else {
84                              return false;
85                      }
86                      str = br.readLine();
87                      String[] komadai_s = str.split(",", 0);
88                      if(komadai_s.length != 7) {
89                              System.out.println("駒台の長さが違います。");
90                              return false;
91                      }
92                      for(int i=0;i<7;i++) {
93                              if(Calc.isNumber(komadai_s[i])) {
94                                      komadai[i] = Integer.parseInt(komadai_s[i]);
95                              }else {
96                                      return false;
97                              }
98                      }
99              } catch (IOException e) {
100                     e.printStackTrace();
101                     return false;
102             }
103             return true;
104     }
105     /**
106      * 駒台にある駒のポイントの合計を返す。
107      * 先手ならプラス、後手ならマイナスで返す。
108      * @return
109      */
110     public int get_all_point_teban() {
111             Koma koma = null;
112             int all_point = 0;
113             for(int i = 1;i<7;i++) {
114                     koma = get_komadai(i);
115                     if(koma != null) {
116                             all_point += koma.get_point_teban() * komadai[i];
117                     }
118             }
119             return (int)(all_point * 1.1);//持ち駒は持っていた方が特
120             //return all_point;
```

```
121         }
122         /**
123          * 駒台にある駒のポイントの合計を返す。
124          * @return
125          */
126         /*
127         public int get_all_point() {
128                 Koma koma = null;
129                 int all_point = 0;
130                 for(int i = 1;i<9;i++) {
131                         koma = get_komadai(i);
132                         if(koma != null) {
133                                 all_point += koma.get_point_teban() * komadai[i];
134                         }
135                 }
136                 return all_point;
137         }
138         */
139         /**
140          * 駒台にある全ての駒のリストを返す。一つの駒は一回しか入らない
141          * @return
142          */
143         public ArrayList<Koma> get_komadai_all_koma(){
144                 ArrayList<Koma> koma_list = new ArrayList<Koma>();
145                 Koma koma = null;
146                 for(int i=1;i<6;i++) {
147                         koma = get_komadai(i);
148                         if(koma != null) {
149                                 koma_list.add(koma);
150                         }
151                 }
152                 return koma_list;
153
154         }
155         public void output_komadi() {
156                 if(teban==1) {
157                         System.out.println("先手の駒台の出力");
158                 }else {
159                         System.out.println("後手の駒台の出力");
160                 }
161                 for(int i=1;i<7;i++) {
162                         if(komadai[i] > 0) {
163                                 System.out.print(change_n_s(i)+":"+komadai[i]+"枚、");
164                         }
165                 }
166                 System.out.println("");
167         }
168
169         /**
170          * 駒をうつ。減らす
171          * @param koma
172          */
173         public void decrease_komadai(Koma koma) {
174                 if(koma.get_teban() != teban){
175                         System.out.println("error: Komadai decrease_komadai teban");
176                         return;//駒台が違う駒台
177                 }
```

```
178                 int n = koma.get_koma_number();
179                 if(komadai[n]<1) {
180                         System.out.println("error: Komadai decrease_komadai not have");
181                         return;
182                 }
183                 komadai[n] -= 1;
184         }
185         /**
186          * 駒をうつ。減らす
187          * @param koma
188          */
189         public void decrease_komadai_koma_number(int koma_number) {
190                 if(koma_number<1 || koma_number>6) {
191                         System.out.println("error: Komadai decrease_komadai_koma_number");
192                         return;
193                 }
194                 if(komadai[koma_number]<1) {
195                         System.out.println("error: Komadai decrease_komadai not have");
196                         return;
197                 }
198                 komadai[koma_number] -= 1;
199         }
200         /**
201          * 駒を駒台に入れる。
202          */
203         public void set_komadai(Koma koma) {
204                 /*
205                 if(koma.get_teban() == teban){
206                         System.out.println("error: Komadai set_komadai teban");
207                         return;//駒台が違う駒台
208                 }
209                 */
210                 ///自由の時も使うから、手番が同じ駒の時もある
211                 int n = koma.get_koma_number();
212                 if(n>10) {
213                         n -= 10;
214                 }
215                 komadai[n] += 1;
216                 //koma_array.add(koma);
217         }
218         /**
219          * 入力 place_b に持ち駒があればそれを返す。
220          * その駒が持ち駒になければ null を返す。
221          * 実際に動かすわけではない
222          * @param place
223          * @return
224          */
225         public Koma get_komadai(int place) {
226                 if(komadai[place] < 1) {
227                         return null;
228                 }
229                 Koma koma = null;
230                 koma = Calc_koma.make_koma(place, teban);
231                 koma.set_motigoma(true);
232                 koma.set_place(teban*10, place);
233                 return koma;
234         }
```

```
235         public int get_koma_number(int place) {
236                 return komadai[place];
237         }
238         public void set_teban(int teban) {
239                 this.teban = teban;
240         }
241         public int get_teban() {
242                 return teban;
243         }
244         private String get_teban_s() {
245                 if(teban==1) {
246                         return "1";
247                 }else if(teban==2) {
248                         return "2";
249                 }
250                 return "error";
251         }
252         private String change_n_s(int n) {
253                 //n:1,2,3,4,5,6,7,8を歩、香、桂、銀、金、玉、角、飛車に変える 10の位があれば成駒にする
254                 String st;
255                 //全角で2文字
256                 switch (n) {
257                         case 0:
258                                 st = "　 ";
259                                 break;
260                         case 1:
261                                 st = " 歩";
262                                 break;
263                         case 2:
264                                 st = " 銀";
265                                 break;
266                         case 3:
267                                 st = " 金";
268                                 break;
269                         case 4:
270                                 st = " 玉";
271                                 break;
272                         case 5:
273                                 st = " 角";
274                                 break;
275                         case 6:
276                                 st = "飛車";
277                                 break;
278                         case 11:
279                                 st = " と";
280                                 break;
281                         case 12:st = "成銀";
282                                 break;
283                         case 15:
284                                 st = " 馬";
285                                 break;
286                         case 16:
287                                 st = " 龍";
288                                 break;
289                         default:
290                                 st = "error";
291                                 break;
```

```
292                     }
293                     return st;
294             }
295             /*
296             private int change_double_int(double dou) {
297                     return (int)dou;
298             }
299             */
300     }
```

・Kyokumen クラス

```
  1  package board;
  2
  3  import java.io.BufferedReader;
  4  import java.io.File;
  5  import java.io.FileReader;
  6  import java.io.FileWriter;
  7  import java.io.IOException;
  8  import java.util.ArrayList;
  9
 10  import koma.Calc_koma;
 11  import koma.Koma;
 12
 13  /**
 14  *
 15  * @author satoshi
 16  * 局面を保存する。盤面と駒台
 17  *  駒 0:なし 1:歩 2:銀 3:金 4:玉 5:角 6:飛車
 18  * 10 の位 0:普通 1:成駒 100 の位 1:先手の駒 2:後手の駒
 19  */
 20
 21  public class Kyokumen {
 22          //盤の状態,11-55 盤の呼び方と同じ
 23          private Komadai sente_komadai;
 24          private Komadai gote_komadai;
 25          private Ban ban;
 26          private int teban;//1:先手 2:後手
 27          private ArrayList<Koma> koma_array;//全てのコマのリスト。駒を探しやすいように. 盤上にある駒だけ
 28
 29          public Kyokumen(Ban ban,Komadai sente_komadai,Komadai gote_komadai,int teban) {
 30                  this.teban = teban;
 31                  this.ban = ban;
 32                  this.sente_komadai = sente_komadai;
 33                  this.gote_komadai = gote_komadai;
 34                  koma_array = new ArrayList<Koma>();
 35                  set_koma_array();
 36                  clone_all_koma();
 37                  decide_all_koma_move_place();///
 38          }
 39          public Kyokumen() {
 40                  //入力がなければ初期局面
 41                  ban = new Ban();
 42                  sente_komadai = new Komadai(1);
 43                  gote_komadai = new Komadai(2);
 44                  teban = 1;
 45                  koma_array = new ArrayList<Koma>();
 46                  set_koma_array();
 47                  decide_all_koma_move_place();///
```

```
48          }
49          public Kyokumen clone() {
50                  return new Kyokumen(ban.clone(),sente_komadai.clone(),gote_komadai.clone(),teban);
51          }
52          /**
53           * @param kyokumen, 比較する局面クラス
54           * @return true:同じ局面,false:違う局面
55           */
56          public boolean equal(Kyokumen kyokumen) {
57                  if(this.teban != kyokumen.get_teban()) {
58                          return false;
59                  }
60                  if(!this.sente_komadai.equal(kyokumen.get_sente_komadai())) {
61                          return false;
62                  }
63                  if(!this.gote_komadai.equal(kyokumen.get_gote_komadai())) {
64                          return false;
65                  }
66                  if(!this.ban.equal(kyokumen.get_ban())) {
67                          return false;
68                  }
69                  return true;
70          }
71          /**
72           * 全ての駒のクローン作って置き換える
73           * 盤上と koma_array
74           */
75          private void clone_all_koma() {
76                  Koma new_koma = null;
77                  ArrayList<Koma> koma_array_new = new ArrayList<Koma>();
78                  for(Koma koma:koma_array) {
79                          new_koma = koma.clone();
80                          koma_array_new.add(new_koma);
81                          set_koma_from_place(new_koma, new_koma.get_place_a(), new_koma.get_place_b());
82                  }
83                  koma_array = koma_array_new;
84          }
85          /**
86           * 自分の駒のリストを返す。
87           * 盤上だけ
88           * @return
89           */
90          public ArrayList<Koma> get_teban_koma_list_ban(int teban){
91                  ArrayList<Koma> my_koma_list = new ArrayList<Koma>();
92                  for(Koma koma:koma_array) {
93                          if(koma.get_teban() == teban) {
94                                  //自分の駒。盤上の駒
95                                  my_koma_list.add(koma);
96                          }
97                  }
98                  return my_koma_list;
99          }
100         /**
101          * 自分の駒のリストを返す。
102          * 持ち駒もリストに入れる。持ち駒の同じ駒は一枚だけ
103          * @return
104          */
```

```
105         public ArrayList<Koma> get_teban_koma_list_all(int teban){
106                 ArrayList<Koma> my_koma_list = get_teban_koma_list_ban(teban);//盤上の駒
107                 //駒台. 先手か後手
108                 for(Koma koma:get_komadai(teban).get_komadai_all_koma()) {
109                         my_koma_list.add(koma);
110                 }
111                 return my_koma_list;
112         }
113         /**
114          * 手番の王様の場所を返す
115          * @return
116          */
117         public int get_place_gyoku(int teban) {
118                 for(Koma koma:koma_array) {
119                         if(koma.get_koma_number()==4) {
120                                 if(koma.get_teban()==teban) {
121                                         return koma.get_place();
122                                 }
123                         }
124                 }
125                 return 0;
126         }
127         /**
128          * 打った駒を受け取って駒台のその駒を減らす。
129          * @param koma
130          */
131         public void decrease_motigoma(Koma koma) {
132                 if(koma.get_teban()==1) {
133                         sente_komadai.decrease_komadai(koma);
134                 }else if(koma.get_teban()==2) {
135                         gote_komadai.decrease_komadai(koma);
136                 }else {
137                         System.out.println("error: Kyokumen decrease_motigoma teban");
138                 }
139         }
140         /**
141          * move もらって前の局面へ
142          * 手番変える
143          * @param move
144          */
145         public void move_back_kyokumen(Move move) {
146                 //動いた駒を元の場所に戻す
147                 Koma move_koma = get_koma_from_place(move.get_after_place_a(), move.get_after_place_b());//
    動いた駒
148                 remove_koma_array(move_koma);
149                 if(move.get_naru()) {
150                         //成っている時
151                         move_koma = move_koma.get_narazukoma();
152                 }
153                 if(move.get_before_place_a()<6) {
154                         //盤上の移動の時
155                         ban.set_banarray(move.get_before_place_a(), move.get_before_place_b(), move_koma);
156                         add_koma_array(move_koma);
157                 }else {
158                         //駒を打った時
159                         move_koma.change_teban();
160                         set_koma_from_place(move_koma,move.get_before_place_a() , move.get_before_place_b());
```

```
161                 }
162                 //動いた後の場所に元あった（なかった）駒を配置
163                 if(move.get_get_koma()==0) {
164                         //とっていない
165                         change_teban();
166                         ban.set_banarray(move.get_after_place_a(), move.get_after_place_b(), null);
167                 }else {
168                         //駒を取っていた。
169                         Koma koma = Calc_koma.make_koma(move.get_get_koma(), teban);
170                         ban.set_banarray(move.get_after_place_a(), move.get_after_place_b(), koma);
171                         int koma_number = move.get_get_koma();
172                         if(koma_number>9) {
173                                 koma_number -= 10;
174                         }
175                         //駒台減らす
176                         if(teban==1) {
177                                 //後手番の駒台
178                                 gote_komadai.decrease_komadai_koma_number(koma_number);
179                         }else {
180                                 sente_komadai.decrease_komadai_koma_number(koma_number);
181                         }
182                         change_teban();
183                         add_koma_array(koma);
184                 }
185                 decide_all_koma_move_place();
186         }
187         /**
188          * move をもらって次の局面へ
189          * before,after
190          * なるかどうかはここで確認する。
191          * 手番変える
192          */
193         public void move_next_kyokumen(Move move) {
194                 int a = move.get_before_place_a();
195                 int b = move.get_before_place_b();
196                 Koma koma = get_koma_from_place(a, b);
197                 if(koma.get_motigoma()) {
198                         //持ち駒を打つとき。
199                         koma.set_motigoma(false);
200                         decrease_motigoma(koma);//駒台減らす。
201                         a = move.get_after_place_a();
202                         b = move.get_after_place_b();
203                         ban.set_banarray(a, b, koma);
204                         add_koma_array(koma);
205                 }else {
206                         //盤上の駒を動かす
207                         ban.set_banarray(a, b, null);//動かす前の場所
208                         a = move.get_after_place_a();
209                         b = move.get_after_place_b();
210                         Koma get_koma = get_banarray(a,b);
211                         if(get_koma != null) {
212                                 //駒を取っていたら駒台へ
213                                 if(teban==1) {
214                                         sente_komadai.set_komadai(get_koma);
215                                 }else {
216                                         gote_komadai.set_komadai(get_koma);
217                                 }
```

```
218                             //取った駒は koma_array から消す
219                             koma_array.remove(get_koma);
220                             //move の get_koma を set
221                             move.set_get_koma(get_koma.get_koma_number());
222                     }
223                     //なれるかの確認，
224                     if(move.get_naru()) {
225                             //成る時
226                             remove_koma_array(koma);
227                             koma = koma.get_narigoma();
228                             add_koma_array(koma);
229                     }
230                     ban.set_banarray(a, b, koma);
231             }
232             change_teban();
233             decide_all_koma_move_place();
234     }
235     private void change_teban() {
236             if(teban==1) {
237                     teban = 2;
238             }else if(teban==2) {
239                     teban = 1;
240             }
241     }
242     public void decide_all_koma_move_place() {
243             for(Koma koma:koma_array) {
244                     koma.move_place_clear();
245                     koma.decide_move_place(this);
246             }
247     }
248     public void add_koma_array(Koma koma) {
249             if(koma.get_place()>55) {
250                     System.out.println("add_koma_array miss");
251                     System.out.println(koma.get_place());
252                     System.out.println(koma.get_place_a());
253                     System.out.println(koma.get_place_b());
254             }
255             koma_array.add(koma);
256     }
257     public void remove_koma_array(Koma koma) {
258             koma_array.remove(koma);
259     }
260     /**
261      * koma_array を作る。
262      * ban，見て作る。
263      * 駒台の駒は入れない
264      */
265     public void set_koma_array() {
266             koma_array = new ArrayList<Koma>();
267             Koma koma = null;
268             //盤
269             for(int i=1;i<6;i++) {
270                     for(int j=0;j<6;j++) {
271                             koma = ban.get_banarray(i, j);
272                             if(koma != null) {
273                                     koma_array.add(koma);
274                                     if(koma.get_place()>55) {
```

```
275                                             System.out.println("set_koma_array miss");///
276                                     }
277                             }
278                     }
279             }
280     }
281     public void output_kyokumen() {
282             //出力する。いい方法あるといい
283             System.out.println("局面を出力します。teban"+teban);
284             //後手の駒台の出力
285             gote_komadai.output_komadi();
286             ban.output_ban();//盤面の出力
287             //先手の駒台の出力
288             sente_komadai.output_komadi();
289             //output_koma_array();//koma_array の出力
290     }
291     /**
292      * file_writer を受け取って局面をそのファイルに保存する。
293      */
294     public void save_kyokumen_from_file_writer(FileWriter file_writer) {
295             try {
296                     //手番
297                     file_writer.write(teban+"\n");
298                     //先手駒台
299                     sente_komadai.save_file(file_writer);
300                     //後手駒台
301                     gote_komadai.save_file(file_writer);
302                     //盤面
303                     ban.save_file(file_writer);
304                     file_writer.write("\n");
305             } catch (IOException e) {
306                     // TODO 自動生成された catch ブロック
307                     e.printStackTrace();
308             }
309     }
310     /**
311      * 局面をファイルに保存する。
312      */
313     public void save_kyokumen(String save_file) {
314             System.out.println("局面を"+save_file+"に保存します。");
315             File file = new File(save_file);
316             try {
317                     FileWriter file_writer = new FileWriter(file);
318                     //手番
319                     file_writer.write(teban+"\n");
320                     //先手駒台
321                     sente_komadai.save_file(file_writer);
322                     //後手駒台
323                     gote_komadai.save_file(file_writer);
324                     //盤面
325                     ban.save_file(file_writer);
326                     file_writer.close();
327             } catch (IOException e) {
328                     // TODO 自動生成された catch ブロック
329                     e.printStackTrace();
330                     System.out.println("保存に失敗しました。");
331             }
```

```
332                 System.out.println("保存されました。");
333         }
334         /**
335          * 局面を読み込む
336          * @return 成功したら true
337          */
338         public boolean read_kyokumen_from_first_kyokumen(BufferedReader br) {
339                 System.out.println("局面を読み込みます。");
340                 try {
341                         //最初は手番を読み込む
342                         String str = br.readLine();
343                         if(str.equals("1")) {
344                                 this.teban = 1;
345                         }else if(str.equals("2")) {
346                                 this.teban = 2;
347                         }else {
348                                 System.out.println("手番の読み込みに失敗しました。");
349                                 return false;
350                         }
351                         if(!sente_komadai.load_file(br)) {
352                                 System.out.println("先手の駒台の読み込みに失敗しました。");
353                                 return false;
354                         }
355                         if(!gote_komadai.load_file(br)) {
356                                 System.out.println("後手の駒台の読み込みに失敗しました。");
357                                 return false;
358                         }
359                         if(!ban.load_file(br)) {
360                                 System.out.println("盤の読み込みに失敗しました。");
361                                 return false;
362                         }
363                         br.readLine();//改行を読む
364                 } catch (IOException e) {
365                         // TODO 自動生成された catch ブロック
366                         e.printStackTrace();
367                 }
368                 return true;
369         }
370         public boolean load_kyokumen(String load_file) {
371                 System.out.println("局面を読み込みます。");
372                 File file = new File(load_file);
373                 try {
374                         FileReader filereader = new FileReader(file);
375                         BufferedReader br = new BufferedReader(filereader);
376                         //最初は手番を読み込む
377                         String str = br.readLine();
378                         if(str.equals("1")) {
379                                 this.teban = 1;
380                         }else if(str.equals("2")) {
381                                 this.teban = 2;
382                         }else {
383                                 System.out.println("手番の読み込みに失敗しました。");
384                                 br.close();
385                                 return false;
386                         }
387                         if(!sente_komadai.load_file(br)) {
388                                 System.out.println("先手の駒台の読み込みに失敗しました。");
```

```
389                             return false;
390                     }
391                     if(!gote_komadai.load_file(br)) {
392                             System.out.println("後手の駒台の読み込みに失敗しました。");
393                             return false;
394                     }
395                     if(!ban.load_file(br)) {
396                             System.out.println("盤の読み込みに失敗しました。");
397                             return false;
398                     }
399                     br.close();
400             } catch (IOException e) {
401                     e.printStackTrace();
402             }
403             return true;
404     }
405     public void output_koma_array() {
406             System.out.println("koma_array を出力します。");
407             for(Koma koma:koma_array) {
408                     koma.output_move_place();
409             }
410     }
411     /**
412      * 入力の場所の駒を返す。何もないときは null
413      * 入力は a,b とか
414      * @return
415      */
416     public Koma get_koma_from_place(int place_a,int place_b) {
417             if(place_a==10) {
418                     Koma koma = sente_komadai.get_komadai(place_b);
419                     return koma;
420             }else if(place_a==20) {
421                     Koma koma = gote_komadai.get_komadai(place_b);
422                     return koma;
423             }else {
424                     return ban.get_banarray(place_a, place_b);
425             }
426     }
427     public Koma get_koma_from_place(int place) {
428             return get_koma_from_place(place/10, place%10);
429     }
430     /**
431      * 場所に駒をセットする。
432      * 盤だけではなく駒台もできるようにする。
433      * @param koma
434      * @param place_a
435      * @param place_b
436      */
437     public void set_koma_from_place(Koma koma,int place_a,int place_b) {
438             if(place_a<6) {
439                     //盤の時
440                     ban.set_banarray(place_a, place_b, koma);
441             }else if(place_a==10){
442                     //先手の駒台
443                     sente_komadai.set_komadai(koma);
444             }else if(place_a==20) {
445                     //後手の駒台
```

```
446                     gote_komadai.set_komadai(koma);
447             }
448     }
449     public Koma get_sente_komadai_koma(int place_b) {
450             return sente_komadai.get_komadai(place_b);
451     }
452     public Koma get_gote_komadai_koma(int place_b) {
453             return gote_komadai.get_komadai(place_b);
454     }
455     public Komadai get_sente_komadai() {
456             return sente_komadai;
457     }
458     public Komadai get_gote_komadai() {
459             return gote_komadai;
460     }
461     public Komadai get_komadai(int teban) {
462             if(teban==1) {
463                     return get_sente_komadai();
464             }else if(teban==2) {
465                     return get_gote_komadai();
466             }
467             System.out.println("error: Kyokumen class get_komadai teban");
468             return null;
469     }
470     public Koma get_banarray(int a,int b) {
471             if (0<a && a<6 && 0<b && b<6) {
472                     return ban.get_banarray(a, b);
473             }
474             return null;
475     }
476     public Koma get_banarray(int place) {
477             int a = place / 10;
478             int b = place % 10;
479             return get_banarray(a, b);
480     }
481     public int get_teban() {
482             return teban;
483     }
484     public ArrayList<Koma> get_koma_array(){
485             return koma_array;
486     }
487     public Ban get_ban() {
488             return ban;
489     }
490     public String get_teban_string() {
491             if(teban==1) {
492                     return "先手";
493             }else if(teban==2) {
494                     return "後手";
495             }
496             return "error:get_teban_string int kyokumen class";
497     }
498     public void set_ban_komadai(Ban ban,Komadai sente,Komadai gote) {
499             this.ban = ban;
500             this.sente_komadai = sente;
501             this.gote_komadai = gote;
502     }
```

```
503  }
```

・Make_board_list クラス

```
  1  package board;
  2
  3  import java.util.ArrayList;
  4
  5  import koma.Koma;
  6
  7  /**
  8   * Board クラスから次に行ける Board クラスのリストを作って返す。
  9   * move クラスを返すこともできる。
 10   *
 11   */
 12  public class Make_board_list {
 13          /**
 14           * 次の手で王手のリストを作る
 15           * @param kyokumen
 16           * @return
 17           */
 18          public static ArrayList<Move> get_next_move_list_only_oute(Kyokumen kyokumen){
 19                  int teban = kyokumen.get_teban();
 20                  //move の list を作る。
 21                  ArrayList<Move> move_list_all = make_move_list(kyokumen,teban);
 22                  if(move_list_all==null) {
 23                          return null;
 24                  }else if(move_list_all.isEmpty()) {
 25                          return null;
 26                  }
 27                  //move_list から反則を除く. 反則かを調べるためには局面を進める必要がある。
 28                  //反則でなくて王手になっている手を探す。
 29                  Kyokumen next_kyokumen = null;
 30                  ArrayList<Move> move_list_oute = new ArrayList<Move>();
 31                  for(Move move:move_list_all) {
 32                          next_kyokumen = kyokumen.clone();
 33                          next_kyokumen.move_next_kyokumen(move);
 34                          if(!Calc_kyokumen_foul.check_foul_all(next_kyokumen, move)) {
 35                                  //反則がない, 王手がかかっているか確認する
 36                                  if(Calc_kyokumen_foul.check_oute(next_kyokumen, next_kyokumen.get_teban())) {
 37                                          move_list_oute.add(move);
 38                                  }
 39                          }
 40                  }
 41                  return move_list_oute;
 42          }
 43          /**
 44           * kyokumen クラスから次に行ける Move クラスのリストを返す
 45           * 評価値を決めたりはしない。反則は除く。
 46           * @param kyokumen
 47           * @return
 48           */
 49          public static ArrayList<Move> get_next_move_list_no_foul(Kyokumen kyokumen) {
 50                  int teban = kyokumen.get_teban();
 51                  //move の list を作る。
 52                  ArrayList<Move> move_list = make_move_list(kyokumen,teban);
 53                  if(move_list==null) {
 54                          return null;
 55                  }else if(move_list.isEmpty()) {
```

```
56                      return null;
57              }
58              //move_list から反則を除く．反則かを調べるためには局面を進める必要がある。
59              Kyokumen next_kyokumen = null;
60              ArrayList<Move> move_list_no_foul = new ArrayList<Move>();
61              for(Move move:move_list) {
62                      next_kyokumen = kyokumen.clone();
63                      next_kyokumen.move_next_kyokumen(move);
64                      if(!Calc_kyokumen_foul.check_foul_all(next_kyokumen, null)) {
65                              //打ち歩詰めを調べる必要はない。?ある?////
66                              //反則がない
67                              move_list_no_foul.add(move);
68                      }
69              }
70              return move_list_no_foul;
71      }
72      /**
73       * 次の手の Board のリストを返す。
74       * 最初は全ての駒の全ての動きを調べる。そのうちいらないのは消していく
75       * 次の手の候補がなければ null 返す
76       * @return
77       */
78      public static ArrayList<Board> get_next_board_list(Board board) {
79              Kyokumen kyokumen = board.get_kyokumen();
80              int teban = kyokumen.get_teban();
81              //move の list を作る。
82              ArrayList<Move> move_list = make_move_list(kyokumen,teban);
83              if(move_list==null) {
84                      return null;
85              }else if(move_list.isEmpty()) {
86                      return null;
87              }
88              //move_list に評価値を加えていく。勝敗が決まる手があれば除いたり、Calc_value クラスにする
89              ArrayList<Board> board_list  = new ArrayList<Board>();
90              Board b = null;
91              for(Move move:move_list) {
92                      b = board.get_move_next_board(move);
93                      board_list.add(b);
94              }
95              //move_list の中でいらないやつを消していく。候補手の手以下にする。
96              //cv_list = select_move(cv_list);
97              return board_list;
98      }
99      /**
100      * kyokumen から move_list を作る。
101      * 全ての駒の全ての動き方のリスト。反則も含まれる
102      * 駒がなれる時は成ると不成の両方
103      * 持ち駒から打つ時は空いている全部の場所
104      * @param my_koma_list
105      * @return
106      */
107     private static ArrayList<Move> make_move_list(Kyokumen kyokumen,int teban){
108             ArrayList<Koma> my_koma_list = kyokumen.get_teban_koma_list_all(teban);
109             ArrayList<Move> move_list = new ArrayList<Move>();
110             Move move = null;
111             for(Koma koma:my_koma_list) {
112                     if(koma.get_motigoma()) {
```

```
113                                     Koma ban_koma = null;
114                                     //駒が持ち駒の時, 盤上の全ての null の場所
115                                     for(int i=1;i<6;i++) {
116                                             for(int j=1;j<6;j++) {
117                                                     ban_koma = kyokumen.get_koma_from_place(i, j);
118                                                     if(ban_koma==null) {
119                                             move = new Move(koma.get_place_a(),koma.get_place_b(),i,j);
120                                                             move_list.add(move);
121                                                     }
122                                             }
123                                     }
124                             }
125                             //盤上の駒の時
126                             for(int place:koma.get_move_place()) {
127                                     move = new Move();
128                                     move.set_place(koma.get_place_a(), koma.get_place_b(), place);
129                                     //反則ないかは確認しない
130                                     move_list.add(move);
131                                     //なれる場合は成るの時も
132                                     if(koma.able_to_naru()) {
133                                             if(move.able_naru(koma.get_teban())) {
134                                                     move = move.clone();
135                                                     move.set_naru(true);
136                                                     move_list.add(move);
137                                             }
138                                     }
139                             }
140                     }
141                     return move_list;
142             }
143
144     }
```

・Move クラス

```
  1  package board;
  2
  3  import java.util.Scanner;
  4
  5  import koma.Koma;
  6  import main.Calc;
  7
  8  /**
  9   * 動いた値を反対からも読めるように保存
 10   */
 11  public class Move{
 12          private int before_place_a;//76 の 7, 駒台なら 10,20
 13          private int before_place_b;//76 の 6
 14          private int after_place_a;
 15          private int after_place_b;
 16          private boolean naru;//true でなった。
 17          private int get_koma;//取っていなければ 0
 18
 19          public Move() {
 20                  before_place_a = 0;
 21                  before_place_b = 0;
 22                  after_place_a = 0;
 23                  after_place_b = 0;
 24                  naru = false;
```

```
25                  get_koma = 0;
26          }
27          public Move(Move move) {
28                  this.before_place_a = move.get_before_place_a();
29                  this.before_place_b = move.get_before_place_b();
30                  this.after_place_a = move.get_after_place_a();
31                  this.after_place_b = move.get_after_place_b();
32                  this.naru = move.get_naru();
33                  this.get_koma = move.get_get_koma();
34          }
35          public Move(int before_a,int before_b,int after_a,int after_b) {
36                  set_before_place_a( before_a);
37                  set_before_place_b(before_b);
38                  set_after_place_a(after_a);
39                  set_after_place_b(after_b);
40                  naru = false;
41                  get_koma = 0;
42          }
43          public Move clone() {
44                  return new Move(this);
45          }
46          /**
47           * move が等しいか判定
48           * @param move
49           * @return 同じなら true
50           */
51          public boolean equals_move(Move move) {
52                  if(this.before_place_a==move.get_before_place_a() &&
53                  this.before_place_b==move.get_before_place_b() &&
54                                  this.after_place_a==move.get_after_place_a() &&
55                                  this.after_place_b==move.get_after_place_b() &&
56                                  this.get_koma==move.get_get_koma()) {
57                          if(this.naru && move.get_naru()) {
58                                  return true;
59                          }
60                          if(!this.naru && !move.get_naru()) {
61                                  return true;
62                          }
63                  }
64                  return false;
65          }
66          /**
67           * 駒がなれるか判断。なれる場合はどっちにするか確認，人間用
68           * 成る時は true 返す
69           * @param koma
70           * @return
71           */
72          public boolean check_naru(Koma koma) {
73                  //持ち駒からは成れない
74                  if(koma.get_motigoma()) {
75                          return false;
76                  }
77                  if(koma.able_to_naru()) {
78                          if(able_naru(koma.get_teban())) {
79                                  //なれる
80                                  Scanner scan = new Scanner(System.in);
81                                  String str;
```

```
82                              while(true) {
83                                      System.out.println("成りますか？  成る:y 不成:n");
84                                      str = scan.next();
85                                      if(str.equals("y")) {
86                                              ///scan.close();
87                                              return true;//koma.get_narigoma();
88                                      }else if(str.equals("n")) {
89                                              ///scan.close();
90                                              return false;
91                                      }
92                                      System.out.println("入力が違います。");
93                              }
94                      }
95              }
96              return false;
97      }
98      /**
99       * なれるかどうかの判断
100      * なれるなら true
101      * 駒の確認も必要
102      * @return
103      */
104     public boolean able_naru(int teban) {
105             if(teban==1) {
106                     if(before_place_b<2 || after_place_b<2) {
107                             return true;
108                     }
109             }else {
110                     if(before_place_b>4 || after_place_b>4) {
111                             return true;
112                     }
113             }
114             return false;
115     }
116     public boolean check_able_naru(Koma koma) {
117             //持ち駒からは成れない
118             if(koma.get_motigoma()) {
119                     return false;
120             }
121             if(koma.able_to_naru()) {
122                     if(koma.get_teban()==1) {
123                             if(before_place_b<2 || after_place_b<2) {
124                                     return true;
125                             }
126                     }else {
127                             if(before_place_b>4 || after_place_b>4) {
128                                     return true;
129                             }
130                     }
131             }
132             return false;
133     }
134     /**
135      * move の出力。テストで使う。///
136      * 棋譜保存の時も使う
137      */
138     public void output_move_test() {
```

```
139                 int n = 0;//不成
140                 if(naru) {
141                         n = 1;//成る
142                 }
143                 System.out.println(before_place_a+","+before_place_b+","+after_place_a+","+after_place_b+","
144                 +n+","+get_koma+","+get_koma);
145                 //System.out.println("");
146         }
147         /**
148          * 棋譜を保存するときのアウトプット
149          */
150         public void output_move_kihu() {
151                 int n = 0;//不成
152                 if(naru) {
153                         n = 1;//なったとき
154                 }
155                 System.out.println(before_place_a+","+before_place_b+","+after_place_a+","+
156                                 after_place_b+","+n+","+get_koma);
157         }
158         /**
159          * 棋譜を保存するときの改行を含む文字列を返す
160          * @return
161          */
162         public String get_move_kihu() {
163                 int n = 0;//不成
164                 if(naru) {
165                         n = 1;//なったとき
166                 }
167                 String ans = before_place_a+","+before_place_b+","+after_place_a+","+
168                                 after_place_b+","+n+","+get_koma + "\n";
169                 return ans;
170         }
171         /**
172          * move の入力。
173          */
174         public void input_move() {
175                 Scanner scan = new Scanner(System.in);
176                 String str;
177                 //before の入力
178                 while(true) {
179                         System.out.println("動かす駒の今の場所を入力してください。");
180                         str = scan.next();
181                         if(Calc.isNumber(str)) {
182                                 int before = Integer.parseInt(str);
183                                 before_place_a = before / 10;
184                                 before_place_b = before % 10;
185                                 break;
186                         }
187                         System.out.println("入力が数字ではありません。");
188                 }
189                 //after の入力
190                 while(true) {
191                         System.out.println("駒の動かす先の場所を入力してください。");
192                         str = scan.next();
193                         if(Calc.isNumber(str)) {
194                                 int after = Integer.parseInt(str);
195                                 after_place_a = after / 10;
```

```
196                             after_place_b = after % 10;
197                             break;
198                     }
199                     System.out.println("入力が数字ではありません。");
200             }
201             ///scan.close();
202     }
203     /**
204      * ファイルから読み込むときに使う。
205      * 一行ずつ
206      * 新しく move クラス作って返す
207      */
208     public Move get_input_move(String str) {
209             String[] input = str.split(",", 0);
210             if(input.length != 6) {
211                     System.out.println("error: Move get_input_move1"+str);
212                     return null;
213             }
214             int[] set_number = new int[6];
215             for(int i=0;i<6;i++) {
216                     if(Calc.isNumber(input[i])) {
217                             set_number[i] = Integer.parseInt(input[i]);
218                     }else {
219                             System.out.println("error: Move get_input_move2"+input[i]);
220                             return null;
221                     }
222             }
223             Move move = new Move();
224             move.set_before_place_a(set_number[0]);
225             move.set_before_place_b(set_number[1]);
226             move.set_after_place_a(set_number[2]);
227             move.set_after_place_b(set_number[3]);
228             if(set_number[4]==1) {
229                     move.set_naru(true);
230             }else if(set_number[4]==0) {
231                     move.set_naru(false);
232             }else {
233                     System.out.println("error: Move get_input_move naru"+set_number[4]);
234                     return null;
235             }
236             move.set_get_koma(set_number[5]);
237             return move;
238     }
239     /**
240      * 入力と同じ場所が after_place か
241      * @return
242      */
243     public boolean match_after_place(int a,int b) {
244             if(a==after_place_a && b==after_place_b) {
245                     return true;
246             }
247             return false;
248     }
249     public boolean match_after_place(int place) {
250             int a = place /10;
251             int b = place % 10;
252             return match_after_place(a, b);
```

```
253         }
254         public int get_before_place_a() {
255                 return before_place_a;
256         }
257         public int get_before_place_b() {
258                 return before_place_b;
259         }
260         public int get_after_place_a() {
261                 return after_place_a;
262         }
263         public int get_after_place_b() {
264                 return after_place_b;
265         }
266         public void set_before_place_a(int before_a) {
267                 if(before_a==10 || before_a==20 || (0<before_a && before_a<6) ) {
268                         this.before_place_a = before_a;
269                 }
270         }
271         public void set_before_place_b(int before_b) {
272                 if(before_b==10 || before_b==20 || (0<before_b && before_b<6) ) {
273                         this.before_place_b = before_b;
274                 }
275         }
276         public void set_place(int before_a,int before_b,int after) {
277                 set_before_place_a( before_a);
278                 set_before_place_b(before_b);
279                 set_after_place(after);
280         }
281         public void set_after_place_a(int after_a) {
282                 if(0<after_a && after_a<6 ) {
283                         this.after_place_a = after_a;
284                 }
285         }
286         public void set_after_place_b(int after_b) {
287                 if(0<after_b && after_b<6 ) {
288                         this.after_place_b = after_b;
289                 }
290         }
291         public void set_after_place(int after_place) {
292                 int a = after_place / 10;
293                 int b = after_place % 10;
294                 set_after_place_a(a);
295                 set_after_place_b(b);
296         }
297         public boolean get_naru() {
298                 return naru;
299         }
300         public int get_get_koma() {
301                 return get_koma;
302         }
303         public void set_naru(boolean naru) {
304                 this.naru = naru;
305         }
306         public void set_get_koma(int koma) {
307                 this.get_koma = koma;
308         }
309         public int get_after_place_mix() {
```

```
310                 return after_place_a*10 + after_place_b;
311         }
312 }
```

・ValueComparator_gote クラス

```
  1 package board;
  2
  3 import java.util.Comparator;
  4
  5 import calc_value.Value;
  6 public class ValueComparator_gote implements Comparator<Board>{
  7
  8         @Override
  9         public int compare(Board b1, Board b2) {
 10                 Value v1 = b1.get_value();
 11                 Value v2 = b2.get_value();
 12                 //勝ちが決まっているならどちらでも良い?, 千日手の時もある
 13                 if(v1.get_determe() && v2.get_determe()) {
 14                         if(v1.get_value() < v2.get_value()) {
 15                                 return 1;
 16                         }else if(v1.get_value() == v2.get_value()) {
 17                                 return 0;
 18                         }
 19                         return -1;
 20                 }
 21                 if(v1.get_determe()) {
 22                         if(v1.get_value() < 0) {
 23                                 return 1;
 24                         }else if(v1.get_value() > 0) {
 25                                 return -1;
 26                         }else {
 27                                 //千日手
 28                                 if(v1.get_value() < v2.get_value()) {
 29                                         return 1;
 30                                 }else if(v1.get_value() == v2.get_value()) {
 31                                         return 0;
 32                                 }
 33                                 return -1;
 34                         }
 35                 }
 36                 if(v2.get_determe()) {
 37                         if(v2.get_value() < 0) {
 38                                 return -1;
 39                         }else if(v2.get_value() > 0) {
 40                                 return 1;
 41                         }else {
 42                                 //千日手
 43                                 if(v1.get_value() < v2.get_value()) {
 44                                         return 1;
 45                                 }else if(v1.get_value() == v2.get_value()) {
 46                                         return 0;
 47                                 }
 48                                 return -1;
 49                         }
 50                 }
 51                 if(v1.get_value() < v2.get_value()) {
 52                         return 1;
 53                 }else if(v1.get_value() == v2.get_value()) {
```

```
                        return 0;
                }
                return -1;
        }
        /**
         * b1 が小さければ true
         * 同じときは気にしていない。
         */
        public static boolean compare_board_gote(Board b1,Board b2) {
                Value v1 = b1.get_value();
                Value v2 = b2.get_value();
                //勝ちが決まっているならどちらでも良い?
                if(v1.get_determe()) {
                        if(v1.get_value() < 0) {
                                return true;
                        }else if(v1.get_value() > 0) {
                                return false;
                        }else {
                                //千日手
                                if(v1.get_value() < v2.get_value()) {
                                        return true;
                                }
                                return false;
                        }
                }
                if(v2.get_determe()) {
                        if(v2.get_value() < 0) {
                                return false;
                        }else if(v2.get_value() > 0) {
                                return true;
                        }else {
                                //千日手
                                if(v1.get_value() < v2.get_value()) {
                                        return true;
                                }
                                return false;
                        }
                }
                if(v1.get_value() < v2.get_value()) {
                        return true;
                }
                return false;
        }

}
```

・ValueComparator_sente クラス

```
package board;

import java.util.Comparator;

import calc_value.Value;

public class ValueComparator_sente implements Comparator<Board> {

        /**
         * b1 がよければ 1,b2 がよければ-1 返す。
         */
```

```
12          @Override
13          public int compare(Board b1, Board b2) {
14                  Value v1 = b1.get_value();
15                  Value v2 = b2.get_value();
16                  //勝ちが決まっているならどちらでも良い?
17                  if(v1.get_determe() && v2.get_determe()) {
18                          if(v1.get_value() < v2.get_value()) {
19                                  return -1;
20                          }if(v1.get_value() == v2.get_value()) {
21                                  return 0;
22                          }
23                          return 1;
24                  }
25                  if(v1.get_determe()) {
26                          if(v1.get_value() > 0) {
27                                  return 1;
28                          }else if(v1.get_value() < 0){
29                                  return -1;
30                          }else {
31                                  //千日手
32                                  if(v1.get_value() < v2.get_value()) {
33                                          return -1;
34                                  }else if(v1.get_value() == v2.get_value()) {
35                                          return 0;
36                                  }
37                                  return 1;
38                          }
39                  }
40                  if(v2.get_determe()) {
41                          if(v2.get_value() > 0) {
42                                  return -1;
43                          }else if(v2.get_value() < 0) {
44                                  return 1;
45                          }else {
46                                  //千日手
47                                  if(v1.get_value() < v2.get_value()) {
48                                          return -1;
49                                  }else if(v1.get_value() == v2.get_value()) {
50                                          return 0;
51                                  }
52                                  return 1;
53                          }
54                  }
55                  if(v1.get_value() < v2.get_value()) {
56                          return -1;
57                  }else if(v1.get_value() == v2.get_value()) {
58                          return 0;
59                  }
60                  return 1;
61          }
62          /**
63           * b1 が大きければ true
64           * 同じときは気にしていない
65           */
66          public static boolean cmpare_board_sente(Board b1,Board b2) {
67                  Value v1 = b1.get_value();
68                  Value v2 = b2.get_value();
```

```
69                  //勝ちが決まっているならどちらでも良い?
70                  if(v1.get_determe()) {
71                          if(v1.get_value() > 0) {
72                                  return true;
73                          }else if(v1.get_value() < 0){
74                                  return false;
75                          }else {
76                                  //千日手
77                                  if(v1.get_value() < v2.get_value()) {
78                                          return false;
79                                  }
80                                  return true;
81                          }
82                  }
83                  if(v2.get_determe()) {
84                          if(v2.get_value() > 0) {
85                                  return false;
86                          }else if(v2.get_value() < 0) {
87                                  return true;
88                          }else {
89                                  //千日手
90                                  if(v1.get_value() < v2.get_value()) {
91                                          return false;
92                                  }
93                                  return true;
94                          }
95                  }
96                  if(v1.get_value() < v2.get_value()) {
97                          return false;
98                  }
99                  return true;
100
101         }
102
103 }
```

・Foul_value クラス

```
1   package calc_value;
2
3   import board.Calc_kyokumen_foul;
4   import board.Kyokumen;
5   import board.Move;
6
7   /**
8    * 反則を選ばないようにするためのクラス
9    * 反則なら value_boolean を true にする。
10   * 王手放置も反則
11   * @author satoshi
12   *
13   */
14  public class Foul_value extends Parent_value {
15
16          public Foul_value(Kyokumen kyokumen) {
17                  super(kyokumen);
18          }
19          /**
20           * 評価値を決める。
21           * value_boolean,value_int を決める
```

```
22          */
23         public void calc_value(Move before_move) {
24                 if(Calc_kyokumen_foul.check_foul_all(get_kyokumen(),before_move)) {
25                         //反則ある
26                         set_value_boolean(true);
27                         if(get_teban()==1) {
28                                 set_value_int(100);
29                         }else if(get_teban()==2) {
30                                 set_value_int(-100);
31                         }else {
32                                 set_value_int(0);
33                                 System.out.println("error: Foul_value class teban"+get_teban());
34                         }
35                 }else {
36                         set_value_boolean(false);
37                         set_value_int(0);//このクラスでは評価値に差はない。
38                 }
39         }
40
41 }
```

・Likely_to_be_token_value クラス

```
1  package calc_value;
2
3  import java.util.ArrayList;
4
5  import board.Kyokumen;
6  import board.Move;
7  import koma.Koma;
8  /**
9   * 取られそうな駒の評価値を入れる。
10  * 手番によって違いがある。？
11  * 自分が相手の駒を取れれば駒の点の 8 割？
12  * 相手が自分の駒を取れれば駒の点の 2 割？
13  *
14  * 相手が自分の駒を取れるかは考えない。自分が相手の駒を取れる時の探索をする。
15  * 取れる場合はとった先の探索をする。相手の局面を評価する。
16  * 相手の局面の評価値の中で最高のものを残す
17  * 相手の局面でも相手が取れる自分の駒を含めた評価を行う。
18  * value_boolean は何も取れる駒がない時 false，取れる駒がある時は true
19  *
20  */
21
22 public class Likely_to_be_token_value extends Parent_value {
23         //private Count_depth count_depth;
24
25         public Likely_to_be_token_value(Kyokumen kyokumen) {
26                 super(kyokumen);
27                 // TODO 自動生成されたコンストラクター・スタブ
28         }
29         /**
30          * 評価値を決める。
31          * value_boolean,value_int を決める
32          */
33         public void calc_value_1() {
34                 set_value_int(0);
35                 set_value_boolean(true);
36                 //自分の駒のリストを作る，盤上だけ
```

```
37                  ArrayList<Koma> my_koma_list = get_kyokumen().get_teban_koma_list_ban(get_teban());
38                  //自分の駒の動けるところに相手の駒があるか確認する。もし駒があればその先の評価値を決める。
39                  Koma koma_enemy = null;
40                  ///Board board_enemy = null;
41                  Move move = null;
42                  Kyokumen kyokumen_enemy = null;
43                  ///Calc_value calc_value = null;
44                  ArrayList<Integer> next_value_list = new ArrayList<Integer>();//次の局面の評価値の
    リスト
45                  ///System.out.println("Likely_to_be_token able to get koma");
46                  ///count_depth.output_depth();
47                  //count_deoth の depth_likely を設定
48                  int depth_likely = 1;//count_depth.get_depth_likely();
49                  ///count_depth.add_depth_likely();
50                  ///count_depth.set_check_likely(false);
51                  for(Koma koma:my_koma_list) {
52                          for(int place:koma.get_move_place()) {
53                                  koma_enemy = get_kyokumen().get_koma_from_place(place);//自分の駒
    が動ける場所に相手の駒があるか
54                                  if(koma_enemy != null) {
55                                          //確認
56                                          //koma.output_test();
57                                          //System.out.println("Likely_to_be_token there are no koma token");///
58                                          //koma.output_move_place();
59                                          //敵の駒を取ることができる。move クラスを作ってその局面の評価値を
    得る,next_value_list に加える。
60                                          move = new Move(koma.get_place_a(),koma.get_place_b(),place/10,place%10);
61                                          kyokumen_enemy = get_kyokumen().clone();
62                                          kyokumen_enemy.move_next_kyokumen(move);
63                                          ///System.out.println("Likely_to_be_token count_depth");
64                                          ///count_depth.output_depth();
65                                          ///calc_value = new Calc_value(kyokumen_enemy,count_depth);
66                                          ///calc_value.set_before_move(move);
67                                          ///calc_value.set_likely_depth(depth);
68                                          ///calc_value.calc_value_present();
69                                          //次の局面の評価値を next_value_list に加える。
70                                          int value = 0;
71                                          next_value_list.add(value);
72                                  }
73                          }
74                  }
75                  //count_depth.set_check_likely(true);
76                  //count_depth.set_depth_likely(depth_likely);
77                  ///System.out.println("Likely_to_be_token able to get koma fin");
78                  if(next_value_list.isEmpty()) {
79                          //System.out.println("Likely_to_be_token there are no koma token");///
80                          set_value_boolean(false);//何もない時
81                          return;
82                  }
83                  //next_value_list の中で最高の評価値が今の局面の評価値より良ければ良さの 9 割？加える。
84                  if(get_teban()==1) {
85                          int max = next_value_list.get(0);
86                          for(int v:next_value_list) {
87                                  if(max<v) {
88                                          max = v;
89                                  }
90                          }
```

```
91                          set_value_int(max);
92                  }else {
93                          int min = next_value_list.get(0);
94                          for(int v:next_value_list) {
95                                  if(min>v) {
96                                          min = v;
97                                  }
98                          }
99                          set_value_int(min);
100                 }
101         }
102         /*
103         public void set_count_depth(Count_depth cd) {
104                 this.count_depth = cd;
105         }
106         */
107
108 }
```

・Loss_koma_value クラス

```
1   package calc_value;
2
3   import board.Kyokumen;
4   import koma.Koma;
5
6   /**
7    * 駒の損得で評価値を決めるクラス。
8    * 駒クラスで自分の得点を決めさせる。駒クラスで駒の働きによって自分の得点を決めるように変える？
9    * 先手の駒ならプラス、後手の駒はマイナス。
10   *
11   */
12  public class Loss_koma_value extends Parent_value {
13
14          public Loss_koma_value(Kyokumen kyokumen) {
15                  super(kyokumen);
16          }
17          /**
18           * 評価値を決める。
19           * value_boolean,value_int を決める
20           */
21          public void calc_value() {
22                  //set_value_boolean(false);//駒の損得では勝敗は決まらない。
23                  int value = 0;
24                  //盤上の駒
25                  for(Koma koma:get_kyokumen().get_koma_array()) {
26                          value += koma.get_point_teban();
27                  }
28                  //駒台の駒
29                  value += get_kyokumen().get_sente_komadai().get_all_point_teban() +
30                  get_kyokumen().get_gote_komadai().get_all_point_teban();
31                  set_value_int(value);
32          }
33
34  }
```

・Move_place_value クラス

```
1   package calc_value;
2
3   import board.Kyokumen;
```

```
4  import koma.Koma;
5
6  /**
7   * 動ける場所の多さで評価値を変える。
8   * 一つの場所で 10000 くらい？
9   *
10  */
11 public class Move_place_value extends Parent_value {
12
13         public Move_place_value(Kyokumen kyokumen) {
14                 super(kyokumen);
15                 // TODO 自動生成されたコンストラクター・スタブ
16         }
17
18         /**
19          * 評価値を決める。
20          * value_boolean,value_int を決める
21          */
22         public void calc_value() {
23                 //value_boolean は変えない
24                 int value = 0;
25                 int move_place_n = 0;
26                 for(Koma koma:get_kyokumen().get_koma_array()) {
27                         move_place_n = koma.get_move_place().size();
28                         if(koma.get_teban()==2) {
29                                 move_place_n = -1 * move_place_n;
30                         }
31                         value += move_place_n;
32                 }
33                 set_value_int(value);
34         }
35
36 }
```

・Parent_value クラス

```
1  package calc_value;
2
3  import board.Kyokumen;
4
5  /**
6   * kyokumen クラスのそれぞれの評価値を決める親クラス
7   * このクラスを継承して評価値を決めるクラスを作る s
8   *
9   */
10 public class Parent_value {
11         private Kyokumen kyokumen;
12         private boolean value_boolean;//勝敗が決まっているときは true
13         private int value_int;
14         public Parent_value(Kyokumen kyokumen) {
15                 this.kyokumen = kyokumen;
16                 calc_value();
17         }
18
19         /**
20          * 評価値を決める。
21          * value_boolean,value_int を決める
22          */
23         public void calc_value() {
```

```
24                  value_boolean = false;
25                  value_int = 0;
26          }
27          /**
28           * 反則や詰み、王手などダメな手やこれしかないなどの手がある時 true にする?
29           * 反則や王手無視などダメな時だけ。
30           * 勝っているときにどうするかは？？勝ちにできたら強い
31           * @return
32           */
33          public boolean get_value_boolean() {
34                  return value_boolean;
35          }
36          /**
37           * int 型で評価値を返す。
38           * @return
39           */
40          public int get_value_int() {
41                  return value_int;
42          }
43          public Kyokumen get_kyokumen() {
44                  return kyokumen;
45          }
46          public int get_teban() {
47                  return kyokumen.get_teban();
48          }
49          public void set_value_int(int value_int) {
50                  this.value_int = value_int;
51          }
52          public void set_value_boolean(boolean value_boolean) {
53                  this.value_boolean = value_boolean;
54          }
55
56  }
```

・Safely_gyoku_value クラス

・Tada_value クラス

```
1   package calc_value;
2
3   import java.util.ArrayList;
4
5   import board.Kyokumen;
6   import board.Move;
7   import koma.Koma;
8   import main.Calc;
9   /**
10   * タダで取れる駒の評価
11   * タダで取れる敵の駒の点数を追加する
12   * 敵の駒の点数を自分の点数にする。
13   *
14   */
15  public class Tada_value extends Parent_value {
16
17          public Tada_value(Kyokumen kyokumen) {
18                  super(kyokumen);
19                  // TODO 自動生成されたコンストラクター・スタブ
20          }
21          /**
```

```
22              * 評価値を決める。
23              * value_boolean,value_int を決める
24              */
25             public void calc_value() {
26                     //タダで取れる駒のリストを作る。
27                     //自分の駒のリストを作る, 盤上だけ
28                     ArrayList<Koma> my_koma_list = get_kyokumen().get_teban_koma_list_ban(get_teban());
29                     //自分の駒の動けるところに相手の駒があるか確認する。もし駒があればその場所に相手の駒が動ける
   か確認する
30                     Koma koma_enemy = null;
31                     Move move = null;
32                     int value = 0;//評価値の合計を作る。
33                     for(Koma koma:my_koma_list) {
34                             for(int place:koma.get_move_place()) {
35                                     koma_enemy = get_kyokumen().get_koma_from_place(place);//自 分 の 駒
   が動ける場所に相手の駒があるか
36                                     if(koma_enemy != null) {
37                                             //自分の駒の動けるところに相手の駒がある。敵の駒を取ることができ
   る。move クラスを作って次の局面にいく。
38                                             move = new Move(koma.get_place_a(),koma.get_place_b(),place/10,place%10);
39                                             if(check_enemy_koma_move(move)) {
40                                                     //相手の駒が効いていない
41                                                     value += koma_enemy.get_point_teban();//取れる駒の
   評価値を加える
42                                             }
43                                     }
44                             }
45                     }
46                     value = value * -1;
47                     set_value_int(value);
48             }
49             /**
50              * 相手の駒が move の移動先に効いているか確認する。
51              * 効いていなければ true を返す
52              * @param move
53              * @return
54              */
55             private boolean check_enemy_koma_move(Move move) {
56                     Kyokumen kyokumen_enemy = get_kyokumen().clone();
57                     kyokumen_enemy.move_next_kyokumen(move);
58                     //相手の駒のリストを作る。
59                     ArrayList<Koma> koma_list_enemy =
60                     kyokumen_enemy.get_teban_koma_list_ban(Calc.change_teban(get_teban()));
61                     //相手の駒の動けるところにその場所があるか
62                     for(Koma koma:koma_list_enemy) {
63                             for(int place:koma.get_move_place()) {
64                                     if(place == move.get_after_place_mix()) {
65                                             return false;
66                                     }
67                             }
68                     }
69                     return true;
70             }
71
72     }
```

・Tumi_check クラス

```
1   package calc_value;
```

```
2
3  import java.util.ArrayList;
4
5  import board.Calc_kyokumen_foul;
6  import board.Kyokumen;
7  import board.Make_board_list;
8  import board.Move;
9
10 /**
11  * 詰みがあるかどうかを局面から判断するクラス。
12  * kyokumen クラスにあった方が良いが長くなってしまうのでこのクラスを作る。
13  *
14  */
15
16 public class Tumi_check {
17         /**
18          *手番の玉がその時詰んでいるかを確認する。0 手読み
19          * @param kyokumen
20          * @return 詰みがある時 true
21          */
22         public static boolean check_tumi_teban(Kyokumen kyokumen) {
23                 if(Calc_kyokumen_foul.check_oute(kyokumen, kyokumen.get_teban())) {
24                         //王手が手番の玉にかかっている。回避できるか調べる。
25                         ArrayList<Move> move_list = Make_board_list.get_next_move_list_no_foul(kyokumen);
26                         if(move_list==null || move_list.isEmpty()) {
27                                 return true;
28                         }
29                 }
30                 return false;
31         }
32         /**
33          * 局面で n 手以下の詰みがあるか調べる
34          * なるべく最短の詰みを見つけられるように横に調べる。
35          * 手に優先順位をつけられるとよい.
36          * n は奇数を想定している。
37          * @param kyokumen 王手のかかっていない局面。詰将棋と同じ
38          * @return 詰みがあれば最初の一手の move を返す。
39          */
40         public static Move check_tumi_n_te(Kyokumen kyokumen,int n) {
41                 if(n<1) {
42                         return null;
43                 }
44                 //次の手で王手の手のリストを作る。
45                 ArrayList<Move> move_list_oute = Make_board_list.get_next_move_list_only_oute(kyokumen);
46                 //どれか一つでも詰んでいるのがあればその move を返す
47                 Kyokumen kyokumen_next = null;
48                 for(Move move:move_list_oute){
49                         kyokumen_next = kyokumen.clone();
50                         kyokumen_next.move_next_kyokumen(move);
51                         //次の局面で玉側が詰されているかを調べる
52                         if(check_tumi_for_gyoku(kyokumen_next, n-1)) {
53                                 //詰んでいる
54                                 return move;
55                         }
56                 }
57                 return null;
58         }
```

```
59          /**
60           * 王手されている局面で n 手以下で詰まされるか調べる。
61           * @param kyokumen n は偶数を想定
62           * @return 詰みがあれば true
63           */
64          public static boolean check_tumi_for_gyoku(Kyokumen kyokumen,int n) {
65                  if(n<0) {
66                          return false;
67                  }
68                  ArrayList<Move> move_list_escape = Make_board_list.get_next_move_list_no_foul(kyokumen);
69                  if(move_list_escape==null || move_list_escape.isEmpty()) {
70                          return true;
71                  }
72                  //全ての逃げ方で詰むかを確認する. 一つでも詰まないのがあれば false
73                  Kyokumen kyokumen_next = null;
74                  for(Move move:move_list_escape) {
75                          kyokumen_next = kyokumen.clone();
76                          kyokumen_next.move_next_kyokumen(move);
77                          if(check_tumi_n_te(kyokumen_next, n-1)==null) {
78                                  //詰まない
79                                  return false;
80                          }
81                  }
82                  //詰まない逃げ方はない
83                  return true;
84          }
85
86  }
```

・Value クラス

```
1   package calc_value;
2
3   import java.util.ArrayList;
4
5   import board.Calc_kyokumen_foul;
6   import board.Kyokumen;
7   import board.Move;
8
9   /**
10   * 評価値のクラス
11   * 現在の局面だけの評価をする
12   *
13   */
14
15  public class Value {
16          private int value;//評価値,//歩特で 1,000,000 くらい？,2147480000max
17          private boolean determe;//勝負が決まっている時 true,
18          /*
19          private static final int max_depth = 3;//最大なんてまで読むか
20          int depth;
21          int value_expect;
22          */
23          public Value(){
24                  determe = false;
25                  value = 0;//
26          }
27          /**
28           * 評価値を計算する。
```

```
29              * 次の手は読まずに現在の局面だけで評価する
30              */
31             public void calc_value_present(Kyokumen kyokumen,ArrayList<Kyokumen> kyokumen_list,Move before_move) {
32                     //反則がないか
33                     Foul_value foul_value = new Foul_value(kyokumen);
34                     foul_value.calc_value(before_move);
35                     determe = foul_value.get_value_boolean();
36                     value = foul_value.get_value_int();
37                     if(determe) {
38                             return;//反則あれば終了
39                     }
40                     if(kyokumen_list!=null) {
41                     //千日手チェック
42                             int sennitite = Calc_kyokumen_foul.check_sennitite(kyokumen_list, kyokumen);
43                             if(sennitite==1) {
44                                     System.out.println("sennitite!");
45                                     set_determe(true);
46                                     set_value(0);
47                                     return;
48                             }else if(sennitite==2) {
49                                     //手番の負け
50                                     set_determe(true);
51                                     set_value(100);
52                                     if(kyokumen.get_teban()==1) {
53                                             set_value(-100);
54                                     }
55                                     return;
56                             }else if(sennitite==3) {
57                                     //手番の勝ち
58                                     set_determe(true);
59                                     set_value(100);
60                                     if(kyokumen.get_teban()==2) {
61                                             set_value(-100);
62                                     }
63                                     return;
64                             }
65                     }
66                     //詰みチェック, 自分の玉に王手がかかっていて 3 手で詰むか
67                     if(Tumi_check.check_tumi_for_gyoku(kyokumen,2)) {
68                             //詰みがある. 手番が負けている。
69                             set_lose(kyokumen.get_teban());
70                             return;
71                     }
72                     //詰みチェック, 相手の玉を詰ますことができるか
73                     if(Tumi_check.check_tumi_n_te(kyokumen, 1)!=null) {
74                             //詰みがある。勝っている
75                             set_win(kyokumen.get_teban());
76                             return;
77                     }
78                     //駒の損得
79                     Loss_koma_value loss_koma_value = new Loss_koma_value(kyokumen);
80                     add_value(loss_koma_value.get_value_int());
81                     //駒の働き、動ける場所
82                     Move_place_value move_place_value = new Move_place_value(kyokumen);
83                     add_value(move_place_value.get_value_int());
84                     //タダで取られる駒
85                     ///System.out.println("calc_value_present Tada_value start:"+get_value_int());
```

```
86                  Tada_value tada_value = new Tada_value(kyokumen);
87                  add_value(tada_value.get_value_int());
88                  ///System.out.println("calc_value_present Tada_value fin:"+get_value_int());
89                  //玉の安全度
90                  Safety_gyoku_value safety_gyoku_value = new Safety_gyoku_value(kyokumen);
91                  add_value(safety_gyoku_value.get_value_int());
92                  ///System.out.println("calc_value_present Safety_gyoku_value fin:"+get_value_int());
93                  //取られそうな駒の評価値を入れる。
94                  //
95          }
96          private void add_value(int add) {
97                  value += add;
98          }
99          public int get_value() {
100                 return value;
101         }
102         public boolean get_determe() {
103                 return determe;
104         }
105         public void set_value(int value) {
106                 this.value = value;
107         }
108         public void set_determe(boolean determe) {
109                 this.determe = determe;
110         }
111         public void set_value_all(Value value) {
112                 this.value = value.get_value();
113                 this.determe = value.get_determe();
114         }
115         /**
116          * 勝ちが決まっている時のその設定をする
117          * @param teban
118          */
119         public void set_win(int teban) {
120                 set_determe(true);
121                 if(teban==1) {
122                         set_value(100);
123                 }else if(teban==2) {
124                         set_value(-100);
125                 }
126         }
127         /**
128          * 負けが決まっている時にその設定をする。
129          * 手番が負け
130          * @param teban
131          */
132         private void set_lose(int teban) {
133                 set_determe(true);
134                 if(teban==1) {
135                         set_value(-100);
136                 }else if(teban==2) {
137                         set_value(100);
138                 }
139         }
140         /**
141          * teban が勝っていれば true
142          * @param teban
```

```
143          * @return
144          */
145         public boolean get_win(int teban) {
146                 if(!determe) {
147                         return false;
148                 }
149                 if(teban==1 && (value > 0)) {
150                         return true;
151                 }
152                 if(teban==2 && (value < 0)) {
153                         return true;
154                 }
155                 return false;
156         }
157
158 }
```

・Calc_Koma クラス

```
  1 package koma;
  2
  3 import main.Calc;
  4
  5 /**
  6  * 駒を作るクラス
  7  * 番号から駒を作る
  8  *
  9  */
 10 public class Calc_koma {
 11         /**
 12          * ファイルから局面を読み込んだ時の読み込み
 13          * @param load_str
 14          */
 15         public static Koma make_load_Koma(String load_str) {
 16                 String[] str = load_str.split(",", 0);
 17                 if(str.length != 4) {
 18                         System.out.println("駒の保存の長さが違います。");
 19                         return null;
 20                 }
 21                 //手番
 22                 int teban = change_teban_int_from_string(str[0]);
 23                 if(teban == 0) {
 24                         return null;
 25                 }
 26                 //place
 27                 int a = -1;
 28                 int b = -1;
 29                 if(Calc.isNumber(str[1])) {
 30                         a = Integer.parseInt(str[1]);
 31                 }else {
 32                         return null;
 33                 }
 34                 if(Calc.isNumber(str[2])) {
 35                         b = Integer.parseInt(str[2]);
 36                 }else {
 37                         return null;
 38                 }
 39                 if(!isBan(a,b)) {
 40                         return null;
```

```
41                  }
42                  int n = -1;
43                  if(Calc.isNumber(str[3])) {
44                          n = Integer.parseInt(str[3]);
45                  }else {
46                          return null;
47                  }
48                  Koma koma = make_koma(n, teban);
49                  if(koma == null) {
50                          return null;
51                  }
52                  koma.set_place(a, b);
53                  return koma;
54          }
55          private static int change_teban_int_from_string(String str) {
56                  if(str.equals("1")) {
57                          return 1;
58                  }else if(str.equals("2")) {
59                          return 2;
60                  }
61                  return 0;
62          }
63          /**
64           * 番号から駒を作る。
65           * @param n
66           * @return
67           */
68          public static Koma make_koma(int n,int teban) {
69                  ///
70                  Koma koma = null;
71                  if(n==1) {
72                          koma = new Hu(teban);
73                  }else if(n==2) {
74                          koma = new Gin(teban);
75                  }else if(n==3) {
76                          koma = new Kin(teban);
77                  }else if(n==4) {
78                          koma = new Gyoku(teban);
79                  }else if(n==5) {
80                          koma = new Kaku(teban);
81                  }else if(n==6) {
82                          koma = new Hisha(teban);
83                  }else if(n==11) {
84                          koma = new To(teban);
85                  }else if(n==12) {
86                          koma = new Narigin(teban);
87                  }else if(n==15) {
88                          koma = new Uma(teban);
89                  }else if(n==16) {
90                          koma = new Ryu(teban);
91                  }
92                  return koma;
93          }
94
95
96          /**
97           * a*10+b が 11-99 の中にあるなら true
```

```
98           * @param a
99           * @param b
100          * @return
101          */
102         public static boolean isBan(int a,int b) {
103                 if(a>0 && a<6 && b>0 && b<6) {
104                         return true;
105                 }else {
106                         return false;
107                 }
108         }
109         /**
110          * a*10+b が 11-99 の中にあるなら true
111          * 入力は 76 とか
112          * @param a
113          * @param b
114          * @return
115          */
116         public boolean isBan(int place) {
117                 int a = place /10;
118                 int b = place % 10;
119                 return isBan(a,b);
120         }
121
122
123 }
```

・Koma クラス

```
1  package koma;
2
3  import java.io.FileWriter;
4  import java.io.IOException;
5  import java.util.ArrayList;
6
7  import board.Kyokumen;
8
9  /*
10  * 駒のクラス。自分の状態を保存する
11  */
12
13 public class Koma {
14
15         private int teban;//どちらの駒か。1:先手  2:後手
16         private int place_a;
17         private int place_b;
18         private int point;//何点の価値があるか，一般的に，動ける場所多いとポイントあがる、いらない？
19         private int point_special;//特別なポイント、攻めに重要とか、守りに重要とか
20         private ArrayList<Integer> move_place;//動ける場所，持ち駒の時は打てる場所,76 とか.place 分けら
   れない？
21         private boolean motigoma;//持ち駒でなければ false
22
23         public Koma(int teban_n) {
24                 teban = teban_n;
25                 move_place = new ArrayList<Integer>();
26                 motigoma = false;
27                 place_a = 0;
28                 place_b = 0;
29                 point = 0;
```

```
30                  point_special = 0;
31          }
32          public Koma clone() {
33                  Koma koma_clone = Calc_koma.make_koma(get_koma_number(), get_teban());
34                  koma_clone.set_place(get_place_a(), get_place_b());
35                  koma_clone.set_point(get_point());
36                  koma_clone.set_point_special(get_point_special());
37                  koma_clone.set_move_place(get_move_place_clone_deep());
38                  koma_clone.set_motigoma(get_motigoma());
39                  return koma_clone;
40          }
41          /**
42           * @param koma 比較する駒
43           * @return true:同じ,false:違う
44           */
45          public boolean equal(Koma koma) {
46                  if(koma==null) {
47                          return false;
48                  }
49                  if(this.teban != koma.get_teban()) {
50                          return false;
51                  }
52                  if(this.place_a != koma.get_place_a()) {
53                          return false;
54                  }
55                  if(this.place_b != koma.get_place_b()) {
56                          return false;
57                  }
58                  if(this.motigoma != koma.get_motigoma()) {
59                          return false;
60                  }
61                  return true;
62          }
63          public void save_file(FileWriter file_writer) {
64                  try {
65                          file_writer.write(teban + "," + place_a + "," + place_b + "," + get_koma_number() + "\n");
66                  } catch (IOException e) {
67                          // TODO 自動生成された catch ブロック
68                          e.printStackTrace();
69                  }
70          }
71          /**
72           * 動けるか。反則にならないか。歩香桂だけ
73           * 反則でなければ false, 反則なら true
74           * @return
75           */
76          public boolean able_to_move() {
77                  return false;
78          }
79          /**
80           * 不成の駒作る
81           * おかしい時は自分を返す
82           * @return
83           */
84          public Koma get_narazukoma() {
85                  if(get_koma_number()>10) {
86                          Koma narazu_koma = Calc_koma.make_koma(get_koma_number()-10, get_teban());
```

```
87                      narazu_koma.set_place(get_place_a(), get_place_b());
88                      return narazu_koma;
89              }
90              System.out.println("error: Koma get_narazukoma not narigoma");
91              return this;
92      }
93      /**
94       * 成り駒を返す。
95       * しっかり成り駒を作る。
96       * @return
97       */
98      public Koma get_narigoma() {
99              int n = get_koma_number();
100             Koma koma = Calc_koma.make_koma(n+10, get_teban());
101             koma.set_place(get_place_a(), get_place_b());
102             return koma;
103     }
104     /**
105      * なれる駒は true を返すようにする。
106      * @return
107      */
108     public boolean able_to_naru() {
109             return false;
110     }
111     /**
112      * String 全角 2 文字で返す。スペースは左にする。
113      * @return
114      */
115     public String get_koma_name() {
116             return "  駒";
117     }
118     private String get_teban_s() {
119             if(teban==1) {
120                     return "↑";
121             }else {
122                     return "↓";
123             }
124     }
125     /**
126      * 全角 2, 半角 1, ↑↓, 半角スペース 1 で出力する。
127      */
128     public void output_koma() {
129             System.out.print(get_koma_name() + get_teban_s() + " ");
130             ///System.out.print(get_koma_name()+get_place() + get_teban_s() + " ");///
131     }
132     /**
133      * テスト用のアウトプット
134      */
135     public void output_test() {
136             System.out.println(get_koma_name() + get_teban_s() + " " +get_place());
137     }
138     public void move_place_clear() {
139             move_place.clear();
140     }
141     /**
142      * 動ける場所を決める
143      * 反則は確認しない
```

```
144          * 継承する子クラスで
145          */
146         public void decide_move_place(Kyokumen kyokumen) {
147                 //
148         }
149         /**
150          * 香車などずっと動ける奴の動き
151          * その方向のマスを move_place に加える。
152          */
153         public void move_place_more(Kyokumen kyokumen,int dif_a,int dif_b) {
154                 int a = get_place_a() + dif_a;
155                 int b = get_place_b() + dif_b;
156                 Koma koma = null;
157                 while(Calc_koma.isBan(a, b)) {
158                         koma = kyokumen.get_banarray(a, b);
159                         if(koma != null) {
160                                 if(koma.get_teban() == get_teban()) {
161                                         //自分の駒がある。
162                                         break;
163                                 }else {
164                                         //相手の駒がある。
165                                         add_move_place(a*10+b);
166                                         break;
167                                 }
168                         }else {
169                                 //前に駒がない
170                                 add_move_place(a*10+b);
171                                 b += dif_b;
172                                 a += dif_a;
173                         }
174                 }
175
176         }
177         /**
178          * place が move_place に含まれるか
179          * @param place
180          * @return
181          */
182         public boolean contain_move_place(int place) {
183                 return move_place.contains(place);
184         }
185         //自分のいる場所、駒の名前、動ける場所を出力
186         public void output_move_place() {
187                 System.out.print(get_koma_name() + get_place_a() + "" + get_place_b()
188                 + "move_place:");
189                 for(int i:move_place) {
190                         System.out.print(i+",");
191                 }
192                 System.out.println("");
193         }
194         public void set_teban(int teban_n) {
195                 teban = teban_n;
196         }
197         public int get_teban() {
198                 return teban;
199         }
200         public void set_place(int a,int b) {
```

```
201                 this.place_a = a;
202                 this.place_b = b;
203         }
204         public void set_point(int point_n) {
205                 point = point_n;
206         }
207         public int get_point() {
208                 return point;
209         }
210         public void set_point_special(int point_special_n) {
211                 point_special = point_special_n;
212         }
213         public int get_point_special() {
214                 return point_special;
215         }
216         public void set_move_place(ArrayList<Integer> move_place) {
217                 this.move_place = move_place;
218         }
219         public  ArrayList<Integer> get_move_place() {
220                 return move_place;
221         }
222         public  ArrayList<Integer> get_move_place_clone_deep() {
223                 ArrayList<Integer> move_place_clone = new ArrayList<Integer>();
224                 for(Integer i:move_place) {
225                         move_place_clone.add(i);
226                 }
227                 return move_place_clone;
228         }
229         public void add_move_place(int place) {
230                 move_place.add(place);
231         }
232         public int get_place_b() {
233                 return place_b;
234         }
235         public int get_place_a() {
236                 return place_a;
237         }
238         public int get_koma_number() {
239                 return 0;
240         }
241         public void set_motigoma(boolean boo) {
242                 this.motigoma = boo;
243         }
244         public boolean get_motigoma() {
245                 return motigoma;
246         }
247         public int get_place() {
248                 return place_a*10+place_b;
249         }
250         public int get_point_teban() {
251                 if(teban==1) {
252                         return get_point();
253                 }else if(teban==2) {
254                         return -1 * get_point();
255                 }
256                 System.out.println("error: Koma get_point_teban teban");
257                 return 0;
```

```
258         }
259         public void change_teban() {
260                 if(teban==1) {
261                         teban = 2;
262                 }else if(teban==2) {
263                         teban = 1;
264                 }else {
265                         System.out.println("error: Koma change_teban teban");
266                 }
267         }
268
269 }
```

・Hu クラス

```
  1 package koma;
  2
  3 import board.Kyokumen;
  4
  5 public class Hu extends Koma {
  6
  7         public Hu(int teban_n) {
  8                 super(teban_n);
  9                 set_point(100);
 10         }
 11
 12         /**
 13          * 動けるか。反則にならないか。歩香桂だけ
 14          * 反則でなければ false, 反則なら true
 15          * @return
 16          */
 17         public boolean able_to_move() {
 18                 if(get_teban()==1 && get_place_b()==1) {
 19                         return true;
 20                 }
 21                 if(get_teban()==2 && get_place_b() == 5) {
 22                         return true;
 23                 }
 24                 return false;
 25         }
 26         /**
 27          * なれる駒は true を返すようにする。
 28          * @return
 29          */
 30         public boolean able_to_naru() {
 31                 return true;
 32         }
 33
 34         public int get_koma_number() {
 35                 return 1;
 36         }
 37         public String get_koma_name() {
 38                 return "  歩";
 39         }
 40         /**
 41          * 動ける場所を決める
 42          * 反則は確認しない
 43          */
 44         public void decide_move_place(Kyokumen kyokumen) {
```

```
45                  Koma koma = null;
46                  int a = get_place_a();
47                  int b = get_place_b();
48                  int difference = 1;
49                  if(get_teban()==1) {
50                          difference = -1;
51                  }
52                  b += difference;
53                  koma = kyokumen.get_banarray(a, b);
54                  if(koma != null) {
55                          if(koma.get_teban() == get_teban()) {
56                                  return;
57                          }
58                  }
59                  add_move_place(a*10+b);
60
61          }
62  }
```

・Kin クラス

```
1   package koma;
2
3   import board.Kyokumen;
4
5   public class Kin extends Koma {
6
7           public Kin(int teban_n) {
8                   super(teban_n);
9                   set_point(900);
10          }
11
12          public int get_koma_number() {
13                  return 3;
14          }
15          public String get_koma_name() {
16                  return "  金";
17          }
18
19          /**
20           * 動ける場所を決める
21           * 反則は確認しない
22           */
23          public void decide_move_place(Kyokumen kyokumen) {
24                  int difference = 1;//先後によって進むか戻るか
25                  if(get_teban()==1) {
26                          difference = -1;//先手の時
27                  }
28                  int a = get_place_a() * 10;
29                  int b = get_place_b();
30                  int b1 = b + difference;//動く先の場所
31                  Calc_koma calc = new Calc_koma();
32                  Koma koma = null;
33                  int[] array = {a+10+b1,a-10+b1,a+10+b,a-10+b,a+b+1,a+b-1};
34                  for(int i:array) {
35                          if(calc.isBan(i)) {
36                                  koma = kyokumen.get_banarray(i);
37                                  if(koma == null) {
38                                          add_move_place(i);
```

```
39                                  }else {
40                                          if(koma.get_teban() != get_teban()) {
41                                                  add_move_place(i);
42                                          }
43                                  }
44                          }
45                  }
46          }
47
48  }
```

・Gin クラス

```
1   package koma;
2
3   import board.Kyokumen;
4
5   public class Gin extends Koma {
6
7           public Gin(int teban_n) {
8                   super(teban_n);
9                   set_point(800);
10          }
11          /**
12           * なれる駒は true を返すようにする。
13           * @return
14           */
15          public boolean able_to_naru() {
16                  return true;
17          }
18
19          public int get_koma_number() {
20                  return 2;
21          }
22          public String get_koma_name() {
23                  return "  銀";
24          }
25          /**
26           * 動ける場所を決める
27           * 反則は確認しない
28           */
29          public void decide_move_place(Kyokumen kyokumen) {
30                  int difference = 1;//先後によって進むか戻るか
31                  if(get_teban()==1) {
32                          difference = -1;//先手の時
33                  }
34                  int a = get_place_a() * 10;
35                  int b = get_place_b();
36                  int b1 = b + difference;//動く先の場所
37                  Calc_koma calc = new Calc_koma();
38                  Koma koma = null;
39                  int[] array = {a+b1,a+10+b+1,a+10+b-1,a-10+b+1,a-10+b-1};
40                  for(int i:array) {
41                          if(calc.isBan(i)) {
42                                  koma = kyokumen.get_banarray(i);
43                                  if(koma == null) {
44                                          add_move_place(i);
45                                  }else {
46                                          if(koma.get_teban() != get_teban()) {
```

```
47                                                      add_move_place(i);
48                                              }
49                                      }
50                              }
51                      }
52              }
53
54      }
```

・Gyoku クラス

```
1   package koma;
2
3   import board.Kyokumen;
4
5   public class Gyoku extends Koma {
6
7           public Gyoku(int teban_n) {
8                   super(teban_n);
9                   set_point(7000000);
10          }
11
12          public int get_koma_number() {
13                  return 4;
14          }
15          public String get_koma_name() {
16                  return "  玉";
17          }
18
19          /**
20           * 動ける場所を決める
21           * 反則は確認しない
22           */
23          public void decide_move_place(Kyokumen kyokumen) {
24                  int a = get_place_a() * 10;
25                  int b = get_place_b();
26                  Calc_koma calc = new Calc_koma();
27                  Koma koma = null;
28                  int[] array = {a+10+b+1,a+10+b,a+10+b-1,a+b+1,a+b-1,a-10+b+1,a-10+b,a-10+b-1};
29                  for(int i:array) {
30                          if(calc.isBan(i)) {
31                                  koma = kyokumen.get_banarray(i);
32                                  if(koma == null) {
33                                          add_move_place(i);
34                                  }else {
35                                          if(koma.get_teban() != get_teban()) {
36                                                  add_move_place(i);
37                                          }
38                                  }
39                          }
40                  }
41          }
42
43  }
```

・Kaku クラス

```
1   package koma;
2
3   import board.Kyokumen;
4
```

```
 5  public class Kaku extends Koma {
 6
 7          public Kaku(int teban_n) {
 8                  super(teban_n);
 9                  set_point(1300);
10          }
11          /**
12           * なれる駒は true を返すようにする。
13           * @return
14           */
15          public boolean able_to_naru() {
16                  return true;
17          }
18
19          public int get_koma_number() {
20                  return 5;
21          }
22          public String get_koma_name() {
23                  return "  角";
24          }
25          /**
26           * 動ける場所を決める
27           * 反則は確認しない
28           */
29          public void decide_move_place(Kyokumen kyokumen) {
30                  move_place_more(kyokumen, 1, 1);
31                  move_place_more(kyokumen, 1, -1);
32                  move_place_more(kyokumen, -1, 1);
33                  move_place_more(kyokumen, -1, -1);
34          }
35
36  }
```

・Hisha クラス

```
 1  package koma;
 2
 3  import board.Kyokumen;
 4
 5  public class Hisha extends Koma {
 6
 7          public Hisha(int teban_n) {
 8                  super(teban_n);
 9                  set_point(1670);
10          }
11
12          /**
13           * なれる駒は true を返すようにする。
14           * @return
15           */
16          public boolean able_to_naru() {
17                  return true;
18          }
19          public int get_koma_number() {
20                  return 6;
21          }
22          public String get_koma_name() {
23                  return "飛車";
24          }
```

```
25
26          /**
27           * 動ける場所を決める
28           * 反則は確認しない
29           */
30          public void decide_move_place(Kyokumen kyokumen) {
31                  move_place_more(kyokumen, 1, 0);
32                  move_place_more(kyokumen, -1, 0);
33                  move_place_more(kyokumen, 0, 1);
34                  move_place_more(kyokumen, 0, -1);
35          }
36
37  }
```

・To クラス

```
1   package koma;
2
3   public class To extends Kin {
4
5           public To(int teban_n) {
6                   super(teban_n);
7                   set_point(900);
8           }
9           public int get_koma_number() {
10                  return 11;
11          }
12          public String get_koma_name() {
13                  return "  と";
14          }
15
16  }
```

・Narigin クラス

```
1   package koma;
2
3   public class Narigin extends Kin {
4
5           public Narigin(int teban_n) {
6                   super(teban_n);
7                   set_point(900);
8           }
9           public int get_koma_number() {
10                  return 12;
11          }
12          public String get_koma_name() {
13                  return "成銀";
14          }
15
16  }
```

・Ryu クラス

```
1   package koma;
2
3   import board.Kyokumen;
4
5   public class Ryu extends Koma {
6
7           public Ryu(int teban_n) {
8                   super(teban_n);
9                   set_point(2400);
```

```
10          }
11          public int get_koma_number() {
12                  return 16;
13          }
14          public String get_koma_name() {
15                  return "  龍";
16          }
17          /**
18           * 動ける場所を決める
19           * 反則は確認しない
20           */
21          public void decide_move_place(Kyokumen kyokumen) {
22                  move_place_more(kyokumen, 1, 0);
23                  move_place_more(kyokumen, -1, 0);
24                  move_place_more(kyokumen, 0, 1);
25                  move_place_more(kyokumen, 0, -1);
26                  int a = get_place_a() * 10;
27                  int b = get_place_b();
28                  Calc_koma calc = new Calc_koma();
29                  Koma koma = null;
30                  int[] array = {a+10+b+1,a+10+b-1,a-10+b+1,a-10+b-1};
31                  for(int i:array) {
32                          if(calc.isBan(i)) {
33                                  koma = kyokumen.get_banarray(i);
34                                  if(koma == null) {
35                                          add_move_place(i);
36                                  }else {
37                                          if(koma.get_teban() != get_teban()) {
38                                                  add_move_place(i);
39                                          }
40                                  }
41                          }
42                  }
43          }
44
45
46  }
```

・Uma クラス

```
1   package koma;
2
3   import board.Kyokumen;
4
5   public class Uma extends Koma {
6
7           public Uma(int teban_n) {
8                   super(teban_n);
9                   set_point(2000);
10          }
11          public int get_koma_number() {
12                  return 15;
13          }
14          public String get_koma_name() {
15                  return "  馬";
16          }
17          /**
18           * 動ける場所を決める
19           * 反則は確認しない
```

```
20           */
21          public void decide_move_place(Kyokumen kyokumen) {
22                  move_place_more(kyokumen, 1, 1);
23                  move_place_more(kyokumen, 1, -1);
24                  move_place_more(kyokumen, -1, 1);
25                  move_place_more(kyokumen, -1, -1);
26                  int a = get_place_a() * 10;
27                  int b = get_place_b();
28                  Calc_koma calc = new Calc_koma();
29                  Koma koma = null;
30                  int[] array = {a+10+b,a-10+b,a+b+1,a+b-1};
31                  for(int i:array) {
32                          if(calc.isBan(i)) {
33                                  koma = kyokumen.get_banarray(i);
34                                  if(koma == null) {
35                                          add_move_place(i);
36                                  }else {
37                                          if(koma.get_teban() != get_teban()) {
38                                                  add_move_place(i);
39                                          }
40                                  }
41                          }
42                  }
43          }
44
45
46  }
```

・Calc クラス

```
1   package main;
2
3   public class Calc {
4           /*
5            * 文字列が数字かどうかを判断
6            */
7           public static boolean isNumber(String str) {
8                   try {
9                           Integer.parseInt(str);
10                          return true;
11                  }catch(NumberFormatException e) {
12                          return false;
13                  }
14          }
15          public static int change_teban(int teban) {
16                  //手番を変える 1 なら 2,2 なら 1 にする
17                  if (teban==1) {
18                          return 2;
19                  }else if (teban==2) {
20                          return 1;
21                  }
22                  System.out.println("error:Calc change_teban");
23                  return 1;
24          }
25
26  }
```

・Kihu クラス

```
1   package main;
2   import java.io.BufferedReader;
```

```
3  import java.io.File;
4  import java.io.FileReader;
5  import java.io.FileWriter;
6  import java.io.IOException;
7  import java.util.ArrayList;
8
9  import board.Kyokumen;
10 import board.Move;
11
12 /**
13  * 棋譜を保存するクラス
14  *
15  */
16 public class Kihu {
17
18         private ArrayList<Move> kihu_list;
19         private Kyokumen first_kyokumen;
20         private Kyokumen last_kyokumen;//棋譜を読み込んだ時に作る。保存はしない. 対局が終わった時にも作る。

21         public Kihu() {
22                 kihu_list = new ArrayList<Move>();
23         }
24         public Kihu(Kyokumen kyokumen) {
25                 kihu_list = new ArrayList<Move>();
26                 first_kyokumen = kyokumen.clone();
27                 last_kyokumen = null;
28         }
29         public Kihu(ArrayList<Move> kihu_list,Kyokumen first_kyokumen) {
30                 this.kihu_list = kihu_list;
31                 this.first_kyokumen = first_kyokumen;
32                 last_kyokumen = null;
33         }
34         public Kihu clone_kihu() {
35                 ArrayList<Move> kihu_list_clone = new ArrayList<Move>();
36                 for(Move move:kihu_list) {
37                         kihu_list_clone.add(move);
38                 }
39                 return new Kihu(kihu_list_clone,first_kyokumen.clone());
40         }
41         public void make_last_kyokumen() {
42                 Kyokumen last_k = first_kyokumen.clone();
43                 for(Move move:kihu_list) {
44                         last_k.move_next_kyokumen(move);
45                 }
46                 last_kyokumen = last_k;
47         }
48         /**
49          * 最初の局面から保存されている棋譜を読み込む
50          */
51         public void read_kihu_from_first_kyokumen(String input_file) {
52                 System.out.println("棋譜を読み込みます。最初の局面が保存されている。");
53                 kihu_list.clear();
54                 Move move = new Move();
55                 Move input_move = null;
56                 File file = new File(input_file);
57                 try {
58                         FileReader filereader = new FileReader(file);
```

```
59                      BufferedReader br = new BufferedReader(filereader);
60                      first_kyokumen.read_kyokumen_from_first_kyokumen(br);
61                      String str = br.readLine();
62                      while(str != null) {
63                              input_move = move.get_input_move(str);
64                              if(input_move == null) {
65                                      break;
66                              }else {
67                                      kihu_list.add(input_move);
68                              }
69                              str = br.readLine();
70                      }
71                      br.close();
72                      make_last_kyokumen();
73              } catch (IOException e) {
74                      // TODO 自動生成された catch ブロック
75                      e.printStackTrace();
76              }
77      }
78      /**
79       * kihu ファイルを読み込む
80       */
81      public void input_kihu(String input_file){
82              System.out.println("棋譜を読み込みます。");
83              kihu_list.clear();
84              Move move = new Move();
85              Move input_move = null;
86              File file = new File(input_file);
87              try {
88                      FileReader filereader = new FileReader(file);
89                      BufferedReader br = new BufferedReader(filereader);
90                      String str = br.readLine();
91                      while(str != null) {
92                              input_move = move.get_input_move(str);
93                              if(input_move == null) {
94                                      break;
95                              }else {
96                                      kihu_list.add(input_move);
97                              }
98                              str = br.readLine();
99                      }
100                     br.close();
101             } catch (IOException e) {
102                     // TODO 自動生成された catch ブロック
103                     e.printStackTrace();
104             }
105     }
106     /**
107      * 棋譜ファイルを最初の局面から保存する。
108      * 最初の局面を保存して、改行して、棋譜保存する。
109      * @param save_file
110      */
111     public void save_kihu_from_first_kyokumen(String save_file) {
112             System.out.println("棋譜を"+save_file+"に保存します。最初の局面も保存します。");
113             File file = new File(save_file);
114             try {
115                     FileWriter file_writer = new FileWriter(file);
```

```
116                     ///保存する
117                     first_kyokumen.save_kyokumen_from_file_writer(file_writer);
118                     for(Move move:kihu_list) {
119                             file_writer.write(move.get_move_kihu());
120                     }
121                     file_writer.close();
122             } catch (IOException e) {
123                     // TODO 自動生成された catch ブロック
124                     e.printStackTrace();
125                     System.out.println("保存に失敗しました。");
126             }
127     }
128     /**
129      * 棋譜をファイルに保存する。
130      * @param save_file
131      */
132     public void save_kihu_file(String save_file) {
133             System.out.println("棋譜を"+save_file+"に保存します。");
134             File file = new File(save_file);
135             try {
136                     FileWriter file_writer = new FileWriter(file);
137                     for(Move move:kihu_list) {
138                             file_writer.write(move.get_move_kihu());
139                     }
140                     file_writer.close();
141             } catch (IOException e) {
142                     e.printStackTrace();
143                     System.out.println("保存に失敗しました。");
144             }
145             System.out.println("保存されました。");
146     }
147
148     public void output_kihu() {
149             System.out.println("棋譜を出力します。first_kyokumen");
150             first_kyokumen.output_kyokumen();
151             System.out.println("棋譜を出力します。kihu");
152             for(Move move:kihu_list) {
153                     move.output_move_kihu();
154             }
155     }
156     public void add_move(Move move) {
157             kihu_list.add(move);
158     }
159     public ArrayList<Move> get_kihu_list(){
160             return kihu_list;
161     }
162     public Move get_last_move() {
163             if(kihu_list==null || kihu_list.isEmpty()) {
164                     return null;
165             }
166             return kihu_list.get(kihu_list.size()-1);
167     }
168     public void remove_last_move() {
169             kihu_list.remove(kihu_list.size()-1);
170     }
171     public Kyokumen get_first_kyokumen() {
172             return first_kyokumen;
```

```
173         }
174         public Kyokumen get_last_kyokumen() {
175                 return last_kyokumen;
176         }
177         //棋譜を保存する。before*100+after *100 + 駒の番号十の位は成駒なら 1, なる時は 2, 成らずは 3, 普通は
    0, 取った駒？
178 }
```

・Main_swing クラス

```
1  package main;
2
3  import java.util.ArrayList;
4
5  import board.Board;
6  import board.Kyokumen;
7  import swing.Action_kihu_listener;
8  import swing.Action_listener;
9  import swing.Action_listener_menu;
10 import swing.Frame_s;
11 import swing.State_game;
12
13 public class Main_swing{
14
15         public static void main(String[] args) {
16                 Frame_s frame = new Frame_s();
17                 Board board = new Board(new Kyokumen(),new Player(),new ArrayList<Kyokumen>());
18                 State_game state = new State_game(board,frame);
19                 Action_listener action_listener = new Action_listener(board,frame,state);
20                 Action_listener_menu action_listener_menu = new Action_listener_menu(board,frame,state);
21                 Action_kihu_listener action_kihu_listener = new Action_kihu_listener(board, frame, state);
22                 frame.set_action_listener(action_listener, action_listener_menu,action_kihu_listener);
23                 frame.swing2();
24                 state.set_not_gaming();
25         }
26
27
28 }
```

・Main クラス

```
1  package main;
2
3
4  import java.io.File;
5  import java.io.FileWriter;
6  import java.io.IOException;
7  import java.util.ArrayList;
8  import java.util.Scanner;
9
10 import board.Board;
11 import board.Kyokumen;
12 import board.Move;
13
14
15 public class Main {
16
17         /**
18          * @param args
19          */
20         public static void main(String[] args) throws IOException{
```

```
21                  System.out.println("対局を始めます。");
22                  Kyokumen kyokumen = new Kyokumen();
23                  /*
24                  if(!kyokumen.load_kyokumen("kyokumen.txt")) {
25                          return;
26                  }
27                  */
28                  //先手後手のプレイヤー決める
29                  Player player = new Player();
30                  player.decide_player();
31                  //file_write();
32                  ArrayList<Kyokumen> kyokumen_list = new ArrayList<Kyokumen>();
33                  Board board = new Board(kyokumen,player,kyokumen_list);
34                  //board.output_kyokumen();
35                  Kihu kihu = new Kihu();
36                  kihu.input_kihu("inputhu.txt");//棋譜をインプットするとき
37                  //kihu.output_kihu();
38                  kyokumen_list.add(board.get_kyokumen().clone());
39                  for(Move move:kihu.get_kihu_list()) {
40                          board.move_next_kyokumen(move);
41                          kyokumen_list.add(board.get_kyokumen().clone());
42                  }
43                  kyokumen_list.remove(kyokumen_list.size()-1);
44                  //対局中。コンピュータと人間を変えたりするときは対局を中断か投了かしないとできない。
45                  int i = 1;
46                  while(i<1000) {
47                          System.out.println(i+"手目の入力をしてください。"+board.get_teban_s());
48                          //全ての駒の move_place を決める。
49                          board.decide_all_koma_move_place();
50                          board.output_kyokumen();
51                          //反則チェック
52                          if(board.check_foul()) {
53                                  System.out.println(board.get_teban_opposite_s()+"の反則負けです。");
54                                  if(check_yes("戻りますか?")) {
55                                          //まったする
56                                          Move back_move = kihu.get_last_move();
57                                          kihu.remove_last_move();
58                                          board.move_back_kyokumen(back_move);
59                                          continue;
60                                  }else {
61                                          break;
62                                  }
63                          }
64                          //千日手チェック, 同じ局面 4 回目なら千日手で終了
65                          int sennitite = board.check_sennitite(kyokumen_list);
66                          if(sennitite != 0) {
67                                  if(sennitite==1) {
68                                          System.out.println("千日手により引き分けです。");
69                                  }else if(sennitite==2) {
70                                          System.out.println("連続王手による千日手で"+board.get_teban_s()+"の反則負けです。");
71                                  }else if(sennitite==3) {
72                                          System.out.println("連続王手による千日手で"+board.get_teban_opposite_s()+"の反則負けです。");
73                                  }
74                                  if(check_yes("戻りますか?")) {
```

```
75                                      //まったする
76                                      Move back_move = kihu.get_last_move();
77                                      kihu.remove_last_move();
78                                      board.move_back_kyokumen(back_move);
79                                      continue;
80                              }else {
81                                      break;
82                              }
83                      }
84                      //kyokumen_list に局面を追加
85                      kyokumen_list.add(board.get_kyokumen().clone());
86                      //対局終了、中断、投了、まった（二手戻る），次の手の入力と同じところでできるとより良い，コンピュータ同士の対局では邪魔
87                      int check_int = check_interruption();//対局を続けるなら 1、待ったなら 2、投了 3，局面評価 4
88                      //int check_int = 1;
89                      if(check_int==2) {
90                              if(i<2) {
91                                      System.out.println("error: Main back i");
92                                      break;
93                              }
94                              //待った、二手戻る。
95                              Move back_move = kihu.get_last_move();
96                              kihu.remove_last_move();
97                              board.move_back_kyokumen(back_move);
98                              back_move = kihu.get_last_move();
99                              kihu.remove_last_move();
100                             board.move_back_kyokumen(back_move);
101                             i -= 2;
102                     }else if(check_int==1){
103                             //次の手
104                             board = board.go_next_board();
105                             if(board == null) {
106                                     System.out.println("負けました。");
107                                     break;
108                             }
109                             //棋譜を保存する
110                             kihu.add_move(board.get_before_move());
111                             System.out.println("指した手は以下です。");
112                             board.get_before_move().output_move_kihu();///
113                             //kihu.output_kihu();
114                             i++;
115                     }else if(check_int==3) {
116                             //投了
117                             System.out.println("負けました。");
118                             break;
119                     }else if(check_int==4) {
120                             ///test
121                             ///board.test();
122                             ///System.out.println("局面の評価値を出力します。");
123                             ///board.output_value_test();
124                             /*
125                             //局面を評価
126                             System.out.println("評価値を求めます。");
127                             Count_depth cd = new Count_depth();
128                             Calc_value cv = new Calc_value(kyokumen,cd);
129                             //int depth_main = cd.get_depth_main();
```

```
130                                 cv.calc_value_depth();
131                                 //cv.calc_value_present();
132                                 //cd.set_depth_main(depth_main);
133                                 System.out.println("評価値:"+cv.get_value_real());
134                                 */
135                         }else if(check_int==5) {
136                                 //局面保存
137                                 String save_file = "kyokumen.txt";
138                                 ///System.out.println("局面を"+save_file+"に保存します。");
139                                 board.save_kyokumen(save_file);
140                         }
141                 }
142                 //棋譜出力確認
143                 check_output_kihu(kihu);
144                 System.out.println("プログラムを終了します。");
145         }
146         /**
147          * 対局中断なら，対局を続けるなら 1、待ったなら 2、投了 3，局面評価 4
148          * @return
149          */
150         private static int check_interruption() {
151                 Scanner scan = new Scanner(System.in);
152                 String str;
153                 while(true) {
154                         System.out.println("対局を続けますか？ n:次の手，int:中断，back:待った，loss:
    負けました。calc:局面評価，save:局面保存");
155                         str = scan.next();
156                         if(str.equals("n")) {
157                                 return 1;
158                         }else if(str.equals("back")) {
159                                 //待った、行って戻る
160                                 return 2;
161                         }else if(str.equals("int")) {
162                                 ///
163                         }else if(str.equals("loss")) {
164                                 return 3;
165                         }else if(str.equals("calc")) {
166                                 return 4;
167                         }else if(str.equals("save")) {
168                                 return 5;
169                         }
170                         System.out.println("入力が違います。");
171                 }
172                 ///return false;
173         }
174         /**
175          * 棋譜を出力するか確認する
176          * @param kihu
177          */
178         public static void check_output_kihu(Kihu kihu) {
179                 String save_file = "output.txt";
180                 String output_str = "棋譜を出力（保存）しますか？"+save_file;
181                 if(check_yes(output_str)) {
182                         //棋譜保存
183                         //kihu.output_kihu();
184                         kihu.save_kihu_file(save_file);
185                 }
```

```
186         }
187         /**
188          * yes,no を確認して yes なら true
189          * no なら false で返す
190          * 確認の文章を String で受け取る
191          * @return
192          */
193         public static boolean check_yes(String output_str) {
194                 Scanner scan = new Scanner(System.in);
195                 String str;
196                 while(true) {
197                         System.out.println(output_str);
198                         System.out.println("y:yes,n:no を入力してください。");
199                         str = scan.next();
200                         if(str.equals("y")) {
201                                 return true;
202                         }else if(str.equals("n")) {
203                                 return false;
204                         }
205                         System.out.println("入力が正しくありません。");
206                 }
207         }
208         /**
209          * テスト用
210          * @throws IOException
211          */
212         public static void file_write() throws IOException {
213                 File file = new File("testwrite.txt");
214                 FileWriter filewriter = new FileWriter(file);
215                 filewriter.write("hahaha");
216                 filewriter.close();
217         }
218 }
```

・Player クラス

```
1  package main;
2
3  import java.util.Scanner;
4
5  /**
6   * 先手後手が人かコンピュータか保存する
7   *
8   */
9  public class Player {
10         private boolean player_sente;//true ならコンピュータ
11         private boolean player_gote;
12         public Player() {
13                 player_sente = false;
14                 player_gote = false;
15         }
16
17         /**
18          * 先手と後手のプレイヤー決める。
19          * コンピュータか人か。
20          * いつでも変えられるようにする。
21          * @return
22          */
23         public void decide_player(){
```

```
24                  //先手後手のプレイヤーが人かコンピューターか入力させる
25                  Scanner scan = new Scanner(System.in);
26                  String str;
27                  //先手決める
28                  while (true) {
29                          System.out.println("先手番を入力してください。人:human, コンピューター:cp");
30                          str = scan.next();
31                          if(str.equals("human")) {
32                                  player_sente = false;
33                                  break;
34                          }else if(str.equals("cp")) {
35                                  player_sente = true;
36                                  break;
37                          }else {
38                                  System.out.println("入力が正しくありません!");
39                          }
40                  }
41                  //後手決める
42                  while (true) {
43                          System.out.println("後手番を入力してください。人:human, コンピューター:cp");
44                          str = scan.next();
45                          if(str.equals("human")) {
46                                  player_gote = false;
47                                  break;
48                          }else if(str.equals("cp")) {
49                                  player_gote = true;
50                                  break;
51                          }else {
52                                  System.out.println("入力が正しくありません!");
53                          }
54                  }
55                  //scan.close();
56                  output_player();
57          }
58          public void output_player() {
59                  if(player_sente) {
60                          System.out.println("先手番はコンピュータです。");
61                  }else {
62                          System.out.println("先手番は人間です。");
63                  }
64                  if(player_gote) {
65                          System.out.println("後手番はコンピュータです。");
66                  }else {
67                          System.out.println("後手番は人間です。");
68                  }
69          }
70          public String get_player_both() {
71                  return get_player_sente_string() + get_player_gote_string();
72          }
73          private String get_player_sente_string() {
74                  if(player_sente) {
75                          return "先手番はコンピュータです。";
76                  }else {
77                          return "先手番は人間です。";
78                  }
79          }
80          private String get_player_gote_string() {
```

```
81                  if(player_gote) {
82                          return "後手番はコンピュータです。";
83                  }else {
84                          return "後手番は人間です。";
85                  }
86          }
87          public boolean get_player_sente() {
88                  return player_sente;
89          }
90          public boolean get_player_gote() {
91                  return player_gote;
92          }
93          public boolean get_player(int teban) {
94                  if(teban==1) {
95                          return player_sente;
96                  }else if(teban==2) {
97                          return player_gote;
98                  }else {
99                          System.out.println("error:Player get_player");
100                         return false;
101                 }
102         }
103         public void set_player_sente(boolean player_sente) {
104                 this.player_sente = player_sente;
105         }
106         public void set_player_gote(boolean player_gote) {
107                 this.player_gote = player_gote;
108         }
109         /**
110          * cp が含まれていれば true
111          * @return
112          */
113         public boolean contain_cp() {
114                 return player_sente || player_gote;
115         }
116
117 }
```

・Action_kihu_listener クラス

```
1  package swing;
2
3  import java.awt.event.ActionEvent;
4  import java.awt.event.ActionListener;
5
6  import board.Board;
7  import board.Move;
8
9  public class Action_kihu_listener implements ActionListener{
10         private Board board;
11         private Frame_s frame;
12         private State_game state;
13
14         public Action_kihu_listener(Board board,Frame_s frame,State_game state) {
15                 this.board = board;
16                 this.frame = frame;
17                 this.state = state;
18         }
19
```

```
20          @Override
21          public void actionPerformed(ActionEvent e) {
22                  if(e.getActionCommand().equals("最初") && state.check_reading_kihu()) {
23                          System.out.println("back to first kyokumen");
24                          //board = new Board(state.get_kihu().get_first_kyokumen().clone());
25                          board.set_kyokumen(state.get_kihu().get_first_kyokumen().clone());
26                          board.reset_except_kyokumen();
27                          state.get_kihu().get_first_kyokumen().output_kyokumen();
28                          //board.output_kyokumen();
29                          frame.set_kyokumen(board.get_kyokumen());
30                          frame.change_label_teban(board.get_teban_s());
31                          state.set_tesuu_first();
32                  }else if(e.getActionCommand().equals("次へ") && state.check_reading_kihu()) {
33                          Move move = state.get_move_next();
34                          if(move != null) {
35                                  board.move_next_kyokumen(move);
36                                  frame.change_panel(move.get_after_place_a(), move.get_after_place_b(),
37                                  board.get_kyokumen());
38                                  frame.change_panel(move.get_before_place_a(), move.get_before_place_b(),
39                                  board.get_kyokumen());
40                                  if(move.get_get_koma() != 0) {
41                                          //駒を取っていた時
42                                          frame.change_panel((3-board.get_teban())*10, move.get_get_koma()%10,
43                                          board.get_kyokumen());
44                                  }
45                                  frame.change_label_teban(board.get_teban_s());
46                                  state.set_tesuu_add();
47                                  state.get_kihu().get_first_kyokumen().output_kyokumen();///
48                          }
49                  }else if(e.getActionCommand().equals("前へ") && state.check_reading_kihu()) {
50                          //局面を戻す
51                          Move move = state.get_move_back();
52                          if(move != null) {
53                                  board.move_back_kyokumen(move);
54                                  frame.change_panel(move.get_after_place_a(), move.get_after_place_b(),
55                                  board.get_kyokumen());
56                                  frame.change_panel(move.get_before_place_a(), move.get_before_place_b(),
57                                  board.get_kyokumen());
58                                  if(move.get_get_koma() != 0) {
59                                          //駒を取っていた時
60                                          frame.change_panel((board.get_teban())*10, move.get_get_koma()%10,
61                                          board.get_kyokumen());
62                                  }
63                                  frame.change_label_teban(board.get_teban_s());
64                                  state.set_tesuu_decrease();
65                          }
66                  }else if(e.getActionCommand().equals("最後") && state.check_reading_kihu()) {
67                          board.set_kyokumen(state.get_kihu().get_last_kyokumen().clone());
68                          board.reset_except_kyokumen();
69                          frame.set_kyokumen(board.get_kyokumen());
70                          frame.change_label_teban(board.get_teban_s());
71                          state.set_tesuu_end();
72                  }
73
74          }
75
76  }
```

・Action_listener_menu クラス

```
1  package swing;
2
3  import java.awt.event.ActionEvent;
4  import java.awt.event.ActionListener;
5
6  import javax.swing.JFrame;
7  import javax.swing.JOptionPane;
8
9  import board.Board;
10 import board.Free_kyokumen;
11 import board.Move;
12 import calc_value.Tumi_check;
13 import calc_value.Value;
14 import main.Kihu;
15
16 public class Action_listener_menu extends JFrame implements ActionListener{
17         private Board board;
18         private Frame_s frame;
19         private State_game state;//状態を持つ。1:対局中,2:対局していない
20         private static String save_kyokumen_name = "kyokumen_s.txt";//局面を保存する時のファイル名
21         private static String read_kyokumen_name = "kyokumen_r.txt";//局面を読み込む時のファイル名
22         private static String save_kihu_name = "kihu_s.txt"; //棋譜を保存するときのファイル名
23         private static String read_kihu_name = "kihu_r.txt";  //棋譜を読み込むときのファイル名
24
25         public static int dep = 0;
26         public static int routine = 0;
27
28         public Action_listener_menu(Board board,Frame_s frame,State_game state) {
29                 this.board = board;
30                 this.frame = frame;
31                 this.state = state;
32         }
33
34         @Override
35         public void actionPerformed(ActionEvent e) {
36                 if(e.getActionCommand().equals("新規対局") && state.check_not_gaming()) {
37                         int option = JOptionPane.showConfirmDialog(this, "データは失われます。新規対
   局を始めますか？", "最終確認", JOptionPane.YES_NO_OPTION,JOptionPane.PLAIN_MESSAGE);
38                     if (option == JOptionPane.YES_OPTION){
39                                 set_first_kyokumen();
40                     }else if (option == JOptionPane.NO_OPTION){
41                                 System.out.println("新規対局は中止しました");
42                     }
43                 }else if(e.getActionCommand().equals("投了") && state.check_gaming_player()) {
44                         JOptionPane.showMessageDialog(this, board.get_teban_s()+"の負けです。", "
   投了",JOptionPane.PLAIN_MESSAGE);
45                         board.get_kihu().make_last_kyokumen();
46                         state.set_reading_kihu(board.get_kihu().get_kihu_list().size());
47                         frame.set_reading_kihu();
48                 }else if(e.getActionCommand().equals("局面評価")) {
49                         //局面を評価して値を表示. 警告で表示?
50                         //とりあえず現在の局面だけの評価値,n 手先の評価値まで求められるとよい。
51                         Value value = new Value();
52                         value.calc_value_present(board.get_kyokumen(), board.get_kyokumen_list(),
53                         board.get_before_move());
54                         String determine = "す。";
```

```
55                          if(!value.get_determe()) {
56                                  determine = "せん。";
57                          }
58                          JOptionPane.showMessageDialog(this, "探索なしの局面の評価値!\n 勝敗は決まっていま"+determine+"\n 評価値は:"+value.get_value(),"この局面の評価値",JOptionPane.PLAIN_MESSAGE);
59                  }else if(e.getActionCommand().equals("待った")) {
60                          //待ったが押されたら一手戻る。cp と対局する時は 2 手戻る。cp が待ったを押すことはない.before_move がなければ戻らない。
61                          if(board.contain_cp()) {
62                                  matta();
63                          }
64                          matta();
65                  }else if(e.getActionCommand().equals("局面保存")) {
66                          //board にある局面を保存する。
67                          board.save_kyokumen(save_kyokumen_name);
68                          JOptionPane.showMessageDialog(this, save_kyokumen_name+" に局面が保存されました。", "局面保存",JOptionPane.PLAIN_MESSAGE);
69                  }else if(e.getActionCommand().equals("局面読み込み")) {
70                          //局面読み込みは対局していない時しかできない。現在の情報が失われる確認をする。
71                          if(state.check_not_gaming()){
72                                  int option = JOptionPane.showConfirmDialog(this, "データは失われます。局面を読み込みますか？", "最終確認", JOptionPane.YES_NO_OPTION,JOptionPane.PLAIN_MESSAGE);
73                              if (option == JOptionPane.YES_OPTION){
74                                          board.read_kyokumen(read_kyokumen_name);
75                                          frame.set_kyokumen(board.get_kyokumen());
76                              }else if (option == JOptionPane.NO_OPTION){
77                                          System.out.println("局面読み込みは中止しました");
78                              }
79                              state.set_not_gaming();
80                              frame.set_kihu_panel_not_visible();
81                          }
82                  }else if(e.getActionCommand().equals("棋譜保存")) {
83                          //棋譜再生中なら state クラスにある棋譜を保存する。
84                          if(state.check_reading_kihu()) {
85                                  state.get_kihu().save_kihu_from_first_kyokumen(save_kihu_name);
86                          }else {
87                                  board.get_kihu().save_kihu_from_first_kyokumen(save_kihu_name);
88                          }
89                          JOptionPane.showMessageDialog(this, save_kihu_name+" に棋譜が保存されました。", "棋譜保存",JOptionPane.PLAIN_MESSAGE);
90                  }else if(e.getActionCommand().equals("棋譜読み込み")) {
91                          //棋譜読み込みは対局していない時しかできない。現在の情報が失われる確認をする。
92                          if(state.check_not_gaming()){
93                                  int option = JOptionPane.showConfirmDialog(this, "データは失われます。棋譜を読み込みますか？", "最終確認", JOptionPane.YES_NO_OPTION,JOptionPane.PLAIN_MESSAGE);
94                              if (option == JOptionPane.YES_OPTION){
95                                          board.read_kihu(read_kihu_name);
96                                          frame.set_kyokumen(board.get_kyokumen());
97                                          state.set_reading_kihu(0);
98                                          frame.set_kihu_panel_visible();
99                              }else if (option == JOptionPane.NO_OPTION){
100                                         System.out.println("棋譜読み込みは中止しました");
101                             }
102                         }
103                 }else if(e.getActionCommand().equals("現在の局面から対局")) {
104                         //対局していない時なら現在の局面から対局開始。棋譜読み込み中なら棋譜の続きから対局することもできる。
```

```
105                         if(state.check_nothin()) {
106                                 set_this_kyokumen();
107                         }else if(state.check_reading_kihu()) {
108                                 int option = JOptionPane.showConfirmDialog(this, "データは失われま
    す。対局を始めますか？", "最終確認", JOptionPane.YES_NO_OPTION,JOptionPane.PLAIN_MESSAGE);
109                             if (option == JOptionPane.YES_OPTION){
110                                         int option_2 = JOptionPane.showConfirmDialog(this, "棋　譜
    の続きから対局をしますか？", "最終確認", JOptionPane.YES_NO_OPTION,JOptionPane.PLAIN_MESSAGE);
111                                     if (option_2 == JOptionPane.YES_OPTION){
112                                             set_this_kihu();
113                                     }else if (option_2 == JOptionPane.NO_OPTION){
114                                             set_this_kyokumen();
115                                     }
116                             }else if (option == JOptionPane.NO_OPTION){
117                                     System.out.println("対局再開を中止しました");
118                             }
119                         }
120                 }else if(e.getActionCommand().equals("自由")) {
121                         //自由に駒を動かせるようにする
122                         if(state.check_not_gaming()) {
123                                 //自由開始
124                                 int option = JOptionPane.showConfirmDialog(this, "データは失われま
    す。自由局面をはじめますか？", "最終確認", JOptionPane.YES_NO_OPTION,JOptionPane.PLAIN_MESSAGE);
125                             if (option == JOptionPane.YES_OPTION){
126                                         System.out.println("自由局面スタート!");
127                                         Free_kyokumen free_kyokumen = new Free_kyokumen(board.get_kyokumen());
128                                         frame.get_action_listener().set_free_kyokumen(free_kyokumen);
129                                         state.set_free_before();
130                                         frame.set_free_before();
131                             }else if (option == JOptionPane.NO_OPTION){
132                                         System.out.println("自由局面を中止しました");
133                             }
134                         }else if(state.check_free_before() || state.check_free_after()) {
135                                 frame.output_message("自由対局を終わります。", "");
136                                 state.set_not_gaming();
137                                 board.set_first();
138                                 int teban = ask("この局面での手番は先手にしますか？","手番確認");
139                                 if(teban!=2) {
140                                         //エラーは先手番にする
141                                         teban = 1;
142                                 }
143                                 board.set_kyokumen(frame.get_action_listener().get_free_kyokumen().
144                                 get_kyokumen(teban));///
145                                 frame.end_game();
146                         }
147                 }else if(e.getActionCommand().equals("詰みチェック")) {
148                         System.out.println("詰みチェックを始めます。");
149                         Move move = Tumi_check.check_tumi_n_te(board.get_kyokumen().clone(), 7);
150                         if(move!=null) {
151                                 frame.output_message("詰みがあります.\n"+move.get_move_kihu(), "詰
    み");
152                                 System.out.println("詰みがあります。");
153                                 move.output_move_kihu();
154                         }else {
155                                 frame.output_message("詰みはありません。", "詰みなし");
156                                 System.out.println("詰みがありません。");
157                         }
```

```
158                 }
159         }
160         /**
161          * 質問をして yes なら 1,no なら 2
162          * 変なのは 0
163          * @return
164          */
165         private int ask(String question,String title) {
166                 int option = JOptionPane.showConfirmDialog(this, question, title,
167                 JOptionPane.YES_NO_OPTION,JOptionPane.PLAIN_MESSAGE);
168             if (option == JOptionPane.YES_OPTION){
169                         return 1;
170             }else if (option == JOptionPane.NO_OPTION){
171                         return 2;
172             }
173             return 0;
174         }
175         private void matta() {
176                 Move back_move = board.get_kihu().get_last_move();
177                 if(back_move != null) {
178                         board.get_kihu().remove_last_move();
179                         board.delete_last_kyokumen_list();
180                         board.move_back_kyokumen(back_move);
181                         frame.change_panel(back_move.get_after_place_a(), back_move.get_after_place_b(),
182                         board.get_kyokumen());
183                         frame.change_panel(back_move.get_before_place_a(), back_move.get_before_place_b(),
184                         board.get_kyokumen());
185                         if(back_move.get_get_koma() != 0) {
186                                 //駒を取っていた時
187                                 frame.change_panel((board.get_teban())*10, back_move.get_get_koma()%10,
188                                 board.get_kyokumen());
189                         }
190                         frame.change_label_teban(board.get_teban_s());
191                 }
192         }
193         private void set_this_kyokumen() {
194                 board.reset_except_kyokumen();
195                 frame.set_kyokumen(board.get_kyokumen());
196                 //先後決める
197                 if(!set_player()) {
198                         System.out.println("対局は中止しました");
199                         return;
200                 }
201
202                 frame.start_game();
203                 JOptionPane.showMessageDialog(this, board.get_player_both()+"\n よろしくお願いしま
    す。", "対局開始",JOptionPane.PLAIN_MESSAGE);
204                 state.set_gaming(board.get_teban_player());
205                 board.get_kyokumen().set_koma_array();
206                 board.decide_all_koma_move_place();
207         }
208         private void set_this_kihu() {
209                 //棋譜を引き継いで対局再開
210                 board.set_kyokumen(state.get_kihu().get_last_kyokumen());
211                 set_this_kyokumen();
212                 Kihu kihu = state.get_kihu();
213                 board.set_kihu(kihu);
```

```
214                     board.set_before_move(kihu.get_kihu_list().get(kihu.get_kihu_list().size()-1));
215             }
216             /**
217              * 新規対局になった時に,board,frame を初期画面にする。
218              * 先手後手を人かどうかも決める。
219              */
220             private void set_first_kyokumen() {
221                     board.set_first();
222                     frame.set_kyokumen(board.get_kyokumen());
223                     //先後決める
224                     if(!set_player()) {
225                             System.out.println("新規対局は中止しました");
226                             return;
227                     }
228                     //先後決まったので対局開始
229                     frame.start_game();
230                     JOptionPane.showMessageDialog(this, board.get_player_both()+"\n よろしくお願いします。", "対局開始",JOptionPane.PLAIN_MESSAGE);
231                     state.set_gaming(board.get_teban_player());
232             }
233
234             /**
235              * 先後を決める。
236              * @return 決まらなければ false
237              */
238             private boolean set_player() {
239                     String selectvalues[] = {"コンピューター, コンピューター", "人間, コンピューター", "コンピューター, 人間", "人間, 人間"};
240                     String sengo = "";
241                     Object value = JOptionPane.showInputDialog(this, "先手, 後手を決めてください!","先後選択", JOptionPane.INFORMATION_MESSAGE,null, selectvalues, selectvalues[0]);
242                     if (value == null){
243                                     return false;
244                     }else{
245                             sengo = (String)value;
246                             if(sengo.equals("コンピューター, コンピューター")) {
247                                     /*board.set_player_sente(false);
248                                     board.set_player_gote(false);*/
249
250
251                                     board.set_player_sente(true);
252                                     board.set_player_gote(true);
253                                     routine = get_go_next();
254                                     Board.set_routine();
255                             }else if(sengo.equals("人間, コンピューター")) {
256                                     board.set_player_sente(false);
257                                     board.set_player_gote(true);
258                                     dep = get_dep();
259                                     Board.set_depth_max();
260                                     frame.set_com("αβ法");
261                             }else if(sengo.equals("コンピューター, 人間")) {
262                                     board.set_player_sente(true);
263                                     board.set_player_gote(false);
264                                     dep = get_dep();
265                                     Board.set_depth_max();
266                                     frame.set_com("αβ法");
267                             }else if(sengo.equals("人間, 人間")) {
```

```
268                                 /*board.set_player_sente(true);
269                                 board.set_player_gote(true);
270                                 routine = get_go_next();
271                                 Board.set_routine();*/
272
273                                 board.set_player_sente(false);
274                                 board.set_player_gote(false);
275                         }else {
276         System.out.println("error:set_first_kyokumen in Action_listener_menu class no choise = "
277                                 +sengo);
278                                 return false;
279                         }
280                 }
281                 return true;
282         }
283
284
285
286
287
288         /**
289          * 深さを決める。
290          * @return 深さ
291          */
292
293         public int get_dep() {
294                 String selectvalues[] = {"2", "3", "4", "5","6"};
295                 String dep = "";
296                 int n = 0;
297                 Object value = JOptionPane.showInputDialog(this, "深 さ を 決 め て く だ さ い?","深
    さ", JOptionPane.INFORMATION_MESSAGE,null, selectvalues, selectvalues[0]);
298
299                 if (value == null){
300                         System.out.println("取り消されました");
301                     }else{
302                         dep = (String)value;
303                                 if(dep.equals("2")) {
304                                         System.out.println("深さは 2 です");
305                                         n = 2;
306                                         frame.set_dep(" 2 ");
307                                 }else if(dep.equals("3")) {
308                                         System.out.println("深さは 3 です");
309                                         n = 3;
310                                         frame.set_dep(" 3 ");
311                                 }else if(dep.equals("4")) {
312                                         System.out.println("深さは 4 です");
313                                         n = 4;
314                                         frame.set_dep(" 4 ");
315                                 }else if(dep.equals("5")) {
316                                         System.out.println("深さは 5 です");
317                                         n = 5;
318                                         frame.set_dep(" 5 ");
319                                 }else if(dep.equals("6")) {
320                                         System.out.println("深さは 6 です");
321                                         n = 6;
322                                         frame.set_dep(" 6 ");
323                                 }
```

```
324                     }
325                 return n;
326         }
327
328         public static int get_dep2() {
329                 return dep;
330         }
331
332
333         /**
334          * 探索を決める。
335          * n=1 αβ
336          * n=2 詰将棋
337          * @return 探索方法
338          */
339
340         public int get_go_next() {
341                 String selectvalues[] = {"αβ法","先手詰将棋ルーチン","後手詰将棋ルーチン","両手詰将棋
    ルーチン"};
342                 String routine = "";
343                 int n = 0;
344                 Object value = JOptionPane.showInputDialog(this, "探索方法を決めてください?","探
    索", JOptionPane.INFORMATION_MESSAGE,null, selectvalues, selectvalues[0]);
345
346                 if (value == null){
347                         System.out.println("取り消されました");
348                     }else{
349                         routine = (String)value;
350                                 if(routine.equals("先手詰将棋ルーチン")) {
351
352                                         System.out.println("先手詰将棋ルーチン");
353                                         n = 2;
354                                         frame.set_com("先手詰将棋ルーチン");
355                                         dep = get_dep_tumi();
356                                         Board.set_depth_max();
357                                 }else if(routine.equals("後手詰将棋ルーチン")) {
358
359
360
361                                         routine = (String)value;
362                                         if(routine.equals("後手詰将棋ルーチン")) {
363                                                 System.out.println("後手詰将棋ルーチン");
364                                                 n = 3;
365                                                 frame.set_com("後手詰将棋ルーチン");
366                                                 dep = get_dep_tumi();
367                                                 Board.set_depth_max();
368                                 }
369                         }else if(routine.equals("両手詰将棋ルーチン")) {
370
371
372
373                                 routine = (String)value;
374                                 if(routine.equals("両手詰将棋ルーチン")) {
375                                         System.out.println("両手詰将棋ルーチン");
376                                         n = 4;
377                                         frame.set_com("両手詰将棋ルーチン");
378                                         dep = get_dep_tumi();
```

```
379                                         Board.set_depth_max();
380                         }
381                     }else if(routine.equals("αβ法")) {
382
383
384
385                                 routine = (String)value;
386                                 if(routine.equals("αβ法")) {
387                                         System.out.println("αβ法");
388                                         n = 1;
389                                         frame.set_com("αβ法");
390                                         dep = get_dep();
391                                         Board.set_depth_max();
392                         }
393                     }
394                     }
395                 return n;
396         }
397   /*
398   *ルーチンを返す
399   */
400         public static int get_routine() {
401                 return routine;
402         }
403
404
405   /*
406   *詰ルーチンの場合の深さを決める
407   */
408         public int get_dep_tumi() {
409                 String selectvalues[] = {"3", "5", "7","9"};
410                 String dep = "";
411                 int n = 0;
412                 Object value = JOptionPane.showInputDialog(this, "深 さ を 決 め て く だ さ い?","深
    さ", JOptionPane.INFORMATION_MESSAGE,null, selectvalues, selectvalues[0]);
413
414                 if (value == null){
415                         System.out.println("取り消されました");
416                     }else{
417                         dep = (String)value;
418                                 if(dep.equals("3")) {
419                                         System.out.println("深さは 3 です");
420                                         n = 3;
421                                         frame.set_dep(" 3 ");
422                                 }else if(dep.equals("5")) {
423                                         System.out.println("深さは 5 です");
424                                         n = 5;
425                                         frame.set_dep(" 5 ");
426                                 }else if(dep.equals("7")) {
427                                         System.out.println("深さは 7 です");
428                                         n = 7;
429                                         frame.set_dep(" 7 ");
430                                 }else if(dep.equals("9")) {
431                                         System.out.println("深さは 9 です");
432                                         n = 9;
433                                         frame.set_dep(" 9 ");
434                                 }
```

```
435                     }
436                 return n;
437         }
438
439
440 }
```

・Action_listener クラス

```
1  package swing;
2
3  import java.awt.event.ActionEvent;
4  import java.awt.event.ActionListener;
5
6  import javax.swing.JFrame;
7  import javax.swing.JOptionPane;
8
9  import board.Board;
10 import board.Free_kyokumen;
11 import board.Move;
12 import koma.Koma;
13 import main.Calc;
14
15 public class Action_listener extends JFrame implements ActionListener{
16         private Board board;
17         private Frame_s frame;
18         private State_game state;//状態を持つ。1:対局中,2:対局していない
19         private Free_kyokumen free_kyokumen;
20         public Action_listener(Board board,Frame_s frame,State_game state) {
21                 this.board = board;
22                 this.frame = frame;
23                 this.state = state;
24                 free_kyokumen = null;
25         }
26
27         @Override
28         public void actionPerformed(ActionEvent e) {
29                 ///System.out.println(e.getActionCommand()+": player"+board.get_player_both());
30                 if(state.check_chose_before_koma() && check_before_koma(e.getActionCommand())) {
31                         //人間が手番でまだ駒を選択していない。選択された駒は手番の駒である。
32                         int place = Integer.parseInt(e.getActionCommand());
33                         state.set_before_place(place);//before_place を設定して駒を選択した状態にする。

34                         //選択した駒の背景変える
35                         frame.change_button_color(place);
36                 }else if(state.check_chose_after_place()) {
37                         //人間が手番で before_place を選択している。after_place が選択された
38                         if(check_after_place(e.getActionCommand())) {
39                                 //動かせる場所なので動かす。
40                                 int after_place = Integer.parseInt(e.getActionCommand());
41                                 Move move = make_move(state.get_before_place(), after_place);
42                                 board.move_next_kyokumen(move);
43                                 frame.change_panel(move.get_after_place_a(), move.get_after_place_b(),
44                                 board.get_kyokumen());
45                                 frame.change_panel(move.get_before_place_a(), move.get_before_place_b(),
46                                 board.get_kyokumen());
47                                 if(move.get_get_koma() != 0) {
48                                         //駒を取っていた時
49                                         frame.change_panel((3-board.get_teban())*10, move.get_get_koma()%10,
```

```
50                                      board.get_kyokumen());
51                              }
52                              //駒を動かしたら反則がないか確認あればそこで対局終了
53                              if(board.check_foul()) {
54                                      System.out.println(board.get_teban_opposite_s()+"の反則負
  けです。！！");
55                                      JOptionPane.showMessageDialog(this, board.get_teban_opposite_s()+"
  の反則負けです。", "反則",JOptionPane.PLAIN_MESSAGE);
56                                      state.set_not_gaming();
57                                      frame.end_game();
58                              }
59                              int sennitite = board.check_sennitite();
60                              if(sennitite==1) {
61                                      //千日手
62                                      JOptionPane.showMessageDialog(this, "千日手です。", "千日
  手",JOptionPane.PLAIN_MESSAGE);
63                                      state.set_not_gaming();
64                                      frame.end_game();
65                              }else if(sennitite==2) {
66                                      //連続王手の千日手。手番が王手している。
67                                      JOptionPane.showMessageDialog(this, board.get_teban_s()+"
  の反則負けです。", "反則:連続王手",JOptionPane.PLAIN_MESSAGE);
68                                      state.set_not_gaming();
69                                      frame.end_game();
70                              }else if(sennitite==3) {
71                                      //連続王手の千日手。手番が王手されている。
72                                      JOptionPane.showMessageDialog(this, board.get_teban_opposite_s()+"
  の反則負けです。", "反則:連続王手",JOptionPane.PLAIN_MESSAGE);
73                                      state.set_not_gaming();
74                                      frame.end_game();
75                              }
76                              state.set_gaming(board.get_teban_player());
77                      }else {
78                              //動かせない場所を選択したら最初の選択を解除
79                              frame.change_panel(state.get_before_place()/10, state.get_before_place()%10,
80                                       board.get_kyokumen());
81                              state.set_gaming(board.get_teban_player());
82                      }
83              }else if(state.check_free_before() && check_free_before(e.getActionCommand())) {
84                      int place = Integer.parseInt(e.getActionCommand());
85                      state.set_before_place(place);//before_place を設定して駒を選択した状態にする。

86                      //選択した駒の背景変える
87                      frame.change_button_color(place);
88                      state.set_free_after();
89              }else if(state.check_free_after() && check_free_after(e.getActionCommand())) {
90                      ///System.out.println("自由局面 after!"+state.get_before_place());
91                      //駒を動かす。
92                      int place_after = Integer.parseInt(e.getActionCommand());
93                      int place_before = state.get_before_place();
94                      ///System.out.println("自由局面 after place "+place_before+" af: "+place_after);
95                      Koma koma = free_kyokumen.move(place_before, place_after);
96                      frame.change_panel(place_before/10, place_before%10, free_kyokumen);
97                      if(koma==null) {
98                              frame.change_panel(place_after/10, place_after%10, free_kyokumen);
99                      }else {
100                             //駒を駒台に移す時
```

```
101                                 frame.change_panel(place_after/10, koma.get_koma_number()%10, free_kyokumen);
102                         }
103                         state.set_free_before();
104                 }
105         }
106         private boolean check_free_after(String place_s) {
107                 int after_place = 0;
108                 if(Calc.isNumber(place_s)) {
109                         after_place = Integer.parseInt(place_s);
110                 }else {
111                         System.out.println("error: button after place ,free, "+place_s);
112                         return false;
113                 }
114                 //同じ場所をさしていたら動かせる。盤上なら成反転, 駒台なら何もしない
115                 if(after_place == state.get_before_place()) {
116                         return true;
117                 }
118                 if(after_place/100 > 0) {
119                         //駒台を選択している。駒台には必ず動ける?自分の駒台から自分の駒台は無駄だが操作す
    る。
120                         return true;
121                 }
122                 //盤上は駒がなければ動ける
123                 Koma koma = free_kyokumen.get_koma_from_place(after_place);
124                 if(koma == null) {
125                         return true;
126                 }
127                 //盤上に駒があれば正しくない。同じ場所をさしていることはない。
128                 return false;
129         }
130         /**
131          * place が正しいかどうかを判断する。
132          * @param place
133          * @return 正しければ true
134          */
135         private boolean check_free_before(String place_s) {
136                 int place = 0;
137                 if(Calc.isNumber(place_s)) {
138                         place = Integer.parseInt(place_s);
139                 }else {
140                         System.out.println("error: button before place free, "+place_s);
141                         return false;
142                 }
143                 if(free_kyokumen==null) {
144                         System.out.println("error: no free kyokumen , ");
145                 }
146                 Koma koma = free_kyokumen.get_koma_from_place(place);
147                 if(koma == null) {
148                         System.out.println("error: button before place no koma, free,"+place_s);
149                         return false;
150                 }
151                 //駒があれば ok
152                 return true;
153         }
154         /**
155          * Move クラスを作る
156          * @return
```

```
157             */
158             private Move make_move(int before_place,int after_place){
159                     Move move = new Move(before_place/10,before_place%10,after_place/10,after_place%10);
160                     //なれるかどうかを確認する
161                     Koma koma = board.get_kyokumen().get_koma_from_place(state.get_before_place());
162                     if(move.check_able_naru(koma)) {
163                             int option = JOptionPane.showConfirmDialog(this, "成りますか？", "成る確
    認", JOptionPane.YES_NO_OPTION,JOptionPane.PLAIN_MESSAGE);
164                         if (option == JOptionPane.YES_OPTION){
165                                     move.set_naru(true);
166                         }else if (option == JOptionPane.NO_OPTION){
167                                     move.set_naru(false);
168                         }
169                     }
170                     return move;
171             }
172             /**
173              * 駒を選択した後に動かせる場所かどうかをチェック
174              * 動かせる場所なら true
175              * @return
176              */
177             private boolean check_after_place(String place_s) {
178                     int after_place = 0;
179                     if(Calc.isNumber(place_s)) {
180                             after_place = Integer.parseInt(place_s);
181                     }else {
182                             System.out.println("error: button after place , "+place_s);
183                             return false;
184                     }
185                     if(after_place/100 > 0) {
186                             //駒台を選択している。
187                             return false;
188                     }
189                     //before_place の駒をつくる
190                     Koma koma = board.get_kyokumen().get_koma_from_place(state.get_before_place());
191                     if(koma == null) {
192                             return false;
193                     }
194                     if(koma.get_motigoma()) {
195                             //持ち駒ならそこに駒がなければよい
196                             Koma after_koma = board.get_kyokumen().get_koma_from_place(after_place);
197                             if(after_koma==null) {
198                                     return true;
199                             }
200                             return false;
201                     }else {
202                             if(koma.contain_move_place(after_place)) {
203                                     return true;
204                             }
205                     }
206                     return false;
207             }
208             /**
209              * 人間が手番で駒が選択されていない時に選択できる駒かどうかを判断する。
210              * 選択できれば true，選択できない（相手の駒）なら false
211              * @return
212              */
```

```
213         private boolean check_before_koma(String place_s) {
214                 int place = 0;
215                 if(Calc.isNumber(place_s)) {
216                         place = Integer.parseInt(place_s);
217                 }else {
218                         System.out.println("error: button before place , "+place_s);
219                         return false;
220                 }
221                 Koma koma = board.get_kyokumen().get_koma_from_place(place);
222                 if(koma == null) {
223                         System.out.println("error: button before place no koma, "+place_s);
224                         return false;
225                 }else {
226                         if(koma.get_teban()==board.get_teban()) {
227                                 return true;
228                         }
229                 }
230                 return false;
231         }
232         public void set_free_kyokumen(Free_kyokumen free_kyokumen) {
233                 this.free_kyokumen = free_kyokumen;
234         }
235         public Free_kyokumen get_free_kyokumen() {
236                 return free_kyokumen;
237         }
238
239 }
```

・Ban_panel クラス

```
1  package swing;
2
3  import java.awt.BorderLayout;
4  import java.awt.Color;
5  import java.awt.Dimension;
6  import java.awt.GridBagConstraints;
7  import java.awt.GridBagLayout;
8  import java.awt.Image;
9  import java.awt.Window;
10
11 import javax.swing.ImageIcon;
12 import javax.swing.JButton;
13 import javax.swing.JLabel;
14 import javax.swing.JPanel;
15 import javax.swing.border.LineBorder;
16
17 import board.Ban;
18 import board.Kyokumen;
19 import koma.Koma;
20
21 public class Ban_panel extends JPanel{
22         private GridBagLayout grid_bag_layout; //レイアウト
23         private JPanel[][] panel_array; //パネルのリスト
24         private Window him; //ウィンドウ
25
26         public Ban_panel(Window her) {
27                 him = her;
28         }
29
```

```
30          public void change_panel(int a,int b,Kyokumen kyokumen,Action_listener action_listener) {
31                  GridBagLayout gbl = grid_bag_layout;
32                  GridBagConstraints gbc = new GridBagConstraints();
33                  Koma koma = kyokumen.get_koma_from_place(a, b);
34                  if(koma == null) {
35                          JPanel panel_o = panel_array[a-1][b-1];
36                          JLabel l = new JLabel("hoge");
37                          panel_o.add(l,BorderLayout.NORTH);
38                          panel_o.setVisible(false);
39                          //panel.removeAll();
40                          JPanel panel = new JPanel();
41                          //panel.add(l,BorderLayout.NORTH);
42                          panel.setOpaque(false);
43                          panel.setPreferredSize(new Dimension(55,60));
44                          panel.setBorder(new LineBorder(Color.black,1));
45                          gbc.gridx = 4-(a-1);//左右反転
46                          gbc.gridy = b-1;
47                          gbl.setConstraints(panel, gbc);
48                          JButton button = new JButton();
49                          button.setPreferredSize(new Dimension(55,60));
50                          button.setBackground(Color.black);///
51                          //actionlistener 加える
52                          button.addActionListener(action_listener);
53                          button.setActionCommand(""+a+""+b);//盤の番号。76 とか
54                          panel.add(button,BorderLayout.CENTER);
55                          panel.setBackground(Color.lightGray);
56                          ///panel.setOpaque(true);
57                          add(panel);
58                          panel_array[a-1][b-1] = panel;
59                  }else {
60                          JPanel panel_o = panel_array[a-1][b-1];
61                          panel_o.setVisible(false);
62                          //panel.removeAll();
63                          JPanel panel = new JPanel();
64                          panel.setOpaque(false);
65                          panel.setPreferredSize(new Dimension(55,60));
66                          panel.setBorder(new LineBorder(Color.black,1));
67                          gbc.gridx = 4-(a-1);//左右反転
68                          gbc.gridy = b-1;
69                          gbl.setConstraints(panel, gbc);
70                          ///駒の画像を表示させる。
71                          ImageIcon icon = Make_koma_image.make_koma_image(koma.get_koma_number(),
72                          koma.get_teban(), him);
73                          Image image = icon.getImage().getScaledInstance(36, 40, Image.SCALE_DEFAULT);
74                          icon = new ImageIcon(image);
75                          JButton button = new JButton(icon);
76                          button.setContentAreaFilled(false);//背景透明化
77                          button.setBorderPainted(false);
78                          //actionlistener 加える
79                          button.addActionListener(action_listener);
80                          button.setActionCommand(""+a+""+b);//盤の番号。76 とか
81                          panel.add(button,BorderLayout.CENTER);
82                          panel.setBackground(Color.lightGray);
83                          ///panel.setOpaque(true);
84                          add(panel);
85                          panel_array[a-1][b-1] = panel;
86                  }
```

```
87          }
88          public void change_color(int a,int b) {
89                  if(0<a && a<10 && 0<b && b<10) {
90                          JPanel panel = panel_array[a-1][b-1];
91                          if(panel != null) {
92                                  panel.setOpaque(true);
93                          }
94                  }
95          }
96          /*
97          public void change_color_back(int a,int b) {
98                  System.out.println("/////////");
99                  if(0<a && a<10 && 0<b && b<10) {
100                         JPanel panel = panel_array[a-1][b-1];
101                         if(panel != null) {
102                                 panel.setOpaque(false);
103                         }
104                 }
105         }
106         */
107         public void set_ban_panel() {
108                 setBounds(220,50,415,460);//横、縦、幅、高さ
109                 setBackground(new Color(218,165,32));
110                 GridBagLayout gbl = new GridBagLayout();
111                 grid_bag_layout = gbl;
112                 setLayout(gbl);
113                 make_masu(gbl);
114                 panel_array = new JPanel[5][5];//00-44,11 なら 00,23 なら 12
115                 for(int i=0;i<5;i++) {
116                         for(int j=0;j<5;j++) {
117                                 panel_array[i][j] = null;
118                         }
119                 }
120         }
121         private void make_masu(GridBagLayout gbl) {
122                 GridBagConstraints gbc = new GridBagConstraints();
123                 for(int i=1;i<6;i++) {
124                         for(int j=1;j<6;j++) {
125                                 JPanel panel = new JPanel();
126                                 panel.setOpaque(false);
127                                 panel.setPreferredSize(new Dimension(55,60));
128                                 panel.setBorder(new LineBorder(Color.black,1));
129                                 gbc.gridx = i-1;
130                                 gbc.gridy = j-1;
131                                 gbl.setConstraints(panel, gbc);
132                                 add(panel);
133                         }
134                 }
135         }
136         /**
137          * Ban を受け取ってその局面通りに表示させる。
138          * 新しいパネルを一つずつ作って行く
139          * @param ban
140          */
141         public void make_masu_from_ban(Ban ban,Action_listener action_listener) {
142                 removeAll();//前の情報は消す。
143                 set_ban_panel();
```

```
144                 GridBagLayout gbl = grid_bag_layout;
145                 GridBagConstraints gbc = new GridBagConstraints();
146                 for(int i=1;i<6;i++) {
147                         for(int j=1;j<6;j++) {
148                                 Koma koma = ban.get_banarray(i, j);
149                                 if(koma != null) {
150                                         JPanel panel = new JPanel();
151                                         panel.setOpaque(false);
152                                         panel.setPreferredSize(new Dimension(55,60));
153                                         panel.setBorder(new LineBorder(Color.black,1));
154                                         gbc.gridx = 4-(i-1);//左右反転
155                                         gbc.gridy = j-1;
156                                         gbl.setConstraints(panel, gbc);
157                                         ///駒の画像を表示させる。
158                                         ImageIcon icon = Make_koma_image.make_koma_image(koma.get_koma_number(),
159                                          koma.get_teban(), him);
160                                         Image image = icon.getImage().getScaledInstance(36, 40,
161                                         Image.SCALE_DEFAULT);
162                                         icon = new ImageIcon(image);
163                                         JButton button = new JButton(icon);
164                                         button.setContentAreaFilled(false);//背景透明化
165                                         button.setBorderPainted(false);
166                                         //actionlistener 加える
167                                         button.addActionListener(action_listener);
168                                         button.setActionCommand(""+i+""+j);//盤の番号。76 とか
169                                         panel.add(button,BorderLayout.CENTER);
170                                         panel.setBackground(Color.lightGray);
171                                         ///panel.setOpaque(true);
172                                         add(panel);
173                                         panel_array[i-1][j-1] = panel;
174                                 }else {
175                                         //駒のないところにもボタンはつくる
176                                         JPanel panel = new JPanel();
177                                         panel.setOpaque(false);
178                                         panel.setPreferredSize(new Dimension(55,60));
179                                         panel.setBorder(new LineBorder(Color.black,1));
180                                         gbc.gridx = 4-(i-1);//左右反転
181                                         gbc.gridy = j-1;
182                                         gbl.setConstraints(panel, gbc);
183                                         JButton button = new JButton();
184                                         //button.setContentAreaFilled(false);//背景透明化
185                                         //button.setBorderPainted(false);///
186                                         button.setPreferredSize(new Dimension(55,60));
187                                         button.setBackground(Color.black);///
188                                         //actionlistener 加える
189                                         button.addActionListener(action_listener);
190                                         button.setActionCommand(""+i+""+j);//盤の番号。76 とか
191                                         panel.add(button,BorderLayout.CENTER);
192                                         panel.setBackground(Color.lightGray);
193                                         ///panel.setOpaque(true);
194                                         add(panel);
195                                         panel_array[i-1][j-1] = panel;
196                                 }
197                         }
198                 }
199         }
200
```

```
201 }
```

・Frame_s クラス

```
  1 package swing;
  2
  3 import java.awt.Color;
  4
  5 import javax.swing.JFrame;
  6 import javax.swing.JLabel;
  7 import javax.swing.JOptionPane;
  8 import javax.swing.JPanel;
  9 import javax.swing.SpringLayout;
 10 import javax.swing.border.LineBorder;
 11
 12 import board.Board;
 13 import board.Kyokumen;
 14
 15 public class Frame_s extends JFrame{
 16         private Menu_panel panel_menu;
 17         private Ban_panel panel_ban;
 18         private Komadai_panel panel_sente;
 19         private Komadai_panel panel_gote;
 20         private JLabel label_teban;
 21         private Action_listener action_listener;
 22         private Action_listener_menu action_listener_menu;
 23         private Action_kihu_listener action_kihu_listener;
 24         private Kihu_panel panel_kihu;
 25         private JLabel label_com;//探索方法
 26         private JLabel label_dep;//深さ表示
 27         public void set_action_listener(Action_listener action_listener,
 28         Action_listener_menu action_listener_menu,Action_kihu_listener action_kihu_listener) {
 29                 this.action_listener = action_listener;
 30                 this.action_listener_menu = action_listener_menu;
 31                 this.action_kihu_listener = action_kihu_listener;
 32
 33         }
 34         /**
 35          * 次の局面に行く時にこのメソッドを呼んでおく
 36          * 次の局面になってから呼ぶ。盤上の駒は変わっている
 37          */
 38         public void set_next_te(String teban) {
 39                 change_label_teban(teban);
 40         }
 41         public void change_label_teban(String string) {
 42                 label_teban.setText(string);
 43         }
 44
 45         public void change_label_com(String string) {
 46                 label_com.setText(string);
 47         }
 48
 49         public void change_label_dep(String string) {
 50                 label_dep.setText(string);
 51         }
 52
 53         /**
 54          * cp 同士の対局の時に
 55          * 次の手に行くかどうかを確認する。
```

```
56           * @return true:次に行く
57           */
58          public boolean check_next_or_exit() {
59                  int option = JOptionPane.showConfirmDialog(this, "次へ進みますか？", "確認",
60                  JOptionPane.YES_NO_OPTION,JOptionPane.PLAIN_MESSAGE);
61              if (option == JOptionPane.YES_OPTION){
62                          return true;
63              }else if (option == JOptionPane.NO_OPTION){
64                          return false;
65              }
66              return false;
67          }
68          public void output_message(String mes,String title) {
69                  JOptionPane.showMessageDialog(this, mes,title,JOptionPane.PLAIN_MESSAGE);
70          }
71
72          /*
73          *勝敗表示
74          処理時間・手数・平均処理時間も表示
75          */
76          public void output_touryou(String teban) {
77                          if(teban.equals("先手")) {
78                                  System.out.println(teban+"の負けです");
79                                  System.out.println("総処理時間帯"+Board.get_seTime() +"ms");
80                                  System.out.println("総手数"+ (Board.get_count()*2-1) +"回");
81                                  System.out.println("平均処理時間"+ Board.get_seTime()/(Board.get_count()-1) +"ms");
82                                  JOptionPane.showMessageDialog(this, teban+"の 負 け で す 。", "投
  了",JOptionPane.PLAIN_MESSAGE);
83                                  Board.reset_count_time();
84                          }else {
85                          System.out.println(teban+"の負けです");
86                          System.out.println("総処理時間帯"+Board.get_seTime() +"ms");
87                          System.out.println("総手数"+Board.get_count()*2 +"回");
88                          System.out.println("平均処理時間"+ Board.get_seTime()/Board.get_count() +"ms");
89                          JOptionPane.showMessageDialog(this, teban+"の負けです。", "投了",JOptionPane.PLAIN_MESSAGE);
90                          Board.reset_count_time();
91                          }
92                  }
93
94          /**
95           * panel の指定したところだけ変える
96           * place は 76 なら 76
97           * @param place_a
98           * @param place_b
99           */
100         public void change_panel(int place_a,int place_b,Kyokumen kyokumen) {
101                 if(0<place_a && place_a<6) {
102                         //盤上の駒
103                         panel_ban.change_panel(place_a, place_b, kyokumen, action_listener);
104                 }else if(place_a==10) {
105                         //先手の駒台
106                         panel_sente.change_panel(place_b, kyokumen.get_sente_komadai(), action_listener);
107                 }else if(place_a==20) {
108                         panel_gote.change_panel(place_b, kyokumen.get_gote_komadai(), action_listener);
109                 }
110                 this.setVisible(true);
111         }
```

```
112         public void change_button_color(int place) {
113                 //place は 76 なら 76
114                 int a = place / 10;
115                 int b = place % 10;
116                 if(0<a && a<6 && 0<b && b<6) {
117                         //盤上
118                         panel_ban.change_color(a, b);
119                 }
120                 if(a==10) {
121                         panel_sente.change_color(b);
122                 }else if(a==20) {
123                         panel_gote.change_color(b);
124                 }
125         }
126         /*
127         public void change_button_color_back(int place) {
128                 //place は 76 なら 76
129                 int a = place / 10;
130                 int b = place % 10;
131                 if(0<a && a<10 && 0<b && b<10) {
132                         //盤上
133                         panel_ban.change_color_back(a, b);
134                 }
135                 if(a==10) {
136                         panel_sente.change_color_back(b);
137                 }else if(a==20) {
138                         panel_gote.change_color_back(b);
139                 }
140                 this.setVisible(true);
141         }
142         */
143         /**
144          * 対局開始の処理。menu_panel の表示非表示を変える
145          */
146         public void start_game() {
147                 panel_menu.set_gaming();
148                 panel_kihu.set_not_visible();
149         }
150         public void end_game() {
151                 panel_menu.set_not_gaming();
152         }
153         public void set_free_before() {
154                 panel_menu.set_free_before();
155         }
156         public void set_reading_kihu() {
157                 panel_menu.set_not_gaming();
158                 panel_kihu.set_visible();
159         }
160         public void swing2() {
161                 //this.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
162                 this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
163                 this.setTitle("55shogi");
164                 this.setLayout(new SpringLayout());
165                 //this.setLayout(null);
166                 JPanel panel = new JPanel();
167                 panel.setBackground(Color.lightGray);
168                 panel.setLayout(null);
```

```
169                 this.setContentPane(panel);
170                 this.setBounds(0,0,905,550);
171                 make_panel(panel);
172                 this.label_teban = new JLabel("手番:");
173                 label_teban.setBounds(10, 200, 150, 50);//x,y,width,heigh
174                 label_teban.setBorder(new LineBorder(Color.black,2));
175                 panel.add(label_teban);
176                 //探索方法表示
177                 this.label_com = new JLabel("");
178                 label_com.setBounds(650,70,200,50);//x,y,width,heigh
179                 label_com.setBorder(new LineBorder(Color.black,2));
180                 panel.add(label_com);
181                 //深さ表示
182                 this.label_dep = new JLabel("");
183                 label_dep.setBounds(650,130,200,50);//x,y,width,heigh
184                 label_dep.setBorder(new LineBorder(Color.black,2));
185                 panel.add(label_dep);
186
187                 this.setVisible(true);
188         }
189         public void make_panel(JPanel panel) {
190                 panel_ban = new Ban_panel(this);
191                 panel_ban.set_ban_panel();
192                 panel.add(panel_ban);
193                 panel_sente = new Komadai_panel(this);
194                 panel_sente.set_komadai_panel(645, 390);
195                 panel.add(panel_sente);
196                 panel_gote = new Komadai_panel(this);
197                 panel_gote.set_komadai_panel(10, 50);
198                 panel.add(panel_gote);
199                 panel_menu = new Menu_panel();
200                 panel_menu.set_menu_panel(action_listener_menu);
201                 panel.add(panel_menu);
202                 panel_kihu = new Kihu_panel();
203                 panel_kihu.set_kihu_panel(action_kihu_listener);
204                 panel.add(panel_kihu);
205
206                 //初期局面表示する。
207                 Kyokumen kyokumen = new Kyokumen();
208                 panel_ban.make_masu_from_ban(kyokumen.get_ban(),action_listener);
209                 panel_sente.make_komadai_panel(kyokumen.get_sente_komadai(),action_listener);
210                 panel_gote.make_komadai_panel(kyokumen.get_gote_komadai(),action_listener);
211                 panel_menu.set_not_gaming();
212         }
213         /**
214          * 入力の局面を表示する
215          * @param kyokumen
216          */
217         public void set_kyokumen(Kyokumen kyokumen) {
218                 panel_ban.make_masu_from_ban(kyokumen.get_ban(),action_listener);
219                 panel_sente.make_komadai_panel(kyokumen.get_sente_komadai(),action_listener);
220                 panel_gote.make_komadai_panel(kyokumen.get_gote_komadai(),action_listener);
221                 change_label_teban("手番: "+kyokumen.get_teban_string());
222
223                 this.setVisible(true);
224         }
225         public void set_kihu_panel_visible() {
```

```
226                     panel_kihu.set_visible();
227             }
228             public void set_kihu_panel_not_visible() {
229                     panel_kihu.set_not_visible();
230             }
231             public Action_listener get_action_listener() {
232                     return action_listener;
233             }
234
235     /*
236     *探索方法をセットする
237     */
238             public void set_com(String board) {
239                     //change_label_com("<html>探索方法:"+board+"<br>ソートサイズ:<html>");
240                     change_label_com("探索方法:"+board);
241
242                     this.setVisible(true);
243             }
244
245     /*
246     *深さをセットする
247     */
248             public void set_dep(String board) {
249                     //change_label_com("<html>探索方法:"+board+"<br>ソートサイズ:<html>");
250                     change_label_dep("深さ"+board);
251
252                     this.setVisible(true);
253             }
254     }
```

・ImageIcon2 クラス

```
1   package swing;
2
3   import java.awt.Image;
4   import java.awt.Toolkit;
5   import java.awt.Window;
6   import java.awt.image.ImageProducer;
7   import java.net.URL;
8
9   import javax.swing.ImageIcon;
10
11  public class ImageIcon2 extends ImageIcon{
12             public ImageIcon2(String f, Window own){
13               super();
14               Image image;
15               try {
16                 Toolkit tk = own.getToolkit();
17                 URL url = own.getClass().getResource(f);
18                 image = tk.createImage((ImageProducer)url.getContent());
19                 setImage(image);
20               } catch (Exception e) {
21                 System.out.println("Image not Found!");
22               }
23             }
24          }
```

・Kihu_panel クラス

```
1   package swing;
2
```

```
3  import java.awt.Color;
4
5  import javax.swing.JButton;
6  import javax.swing.JPanel;
7
8  /**
9   * 棋譜を再生する時に使うパネル
10  *
11  */
12 public class Kihu_panel extends JPanel{
13         private JButton button_next;
14         private JButton button_back;
15         private JButton button_first;
16         private JButton button_end;
17         public void set_kihu_panel(Action_kihu_listener action_kihu_listener) {
18                 setBackground(Color.darkGray);
19                 setBounds(10,300,145,40);//横、縦、幅、高さ
20                 setLayout(null);
21                 button_first = new JButton("<<");
22                 button_first.addActionListener(action_kihu_listener);
23                 button_first.setActionCommand("最初");//
24                 button_first.setBounds(5, 5, 30, 30);
25                 add(button_first);
26                 button_back = new JButton("<");
27                 button_back.addActionListener(action_kihu_listener);
28                 button_back.setActionCommand("前へ");//戻る
29                 button_back.setBounds(40, 5, 30, 30);
30                 add(button_back);
31                 button_next = new JButton(">");
32                 button_next.addActionListener(action_kihu_listener);
33                 button_next.setActionCommand("次へ");//
34                 button_next.setBounds(75, 5, 30, 30);
35                 add(button_next);
36                 button_end = new JButton(">>");
37                 button_end.addActionListener(action_kihu_listener);
38                 button_end.setActionCommand("最後");//
39                 button_end.setBounds(110, 5, 30, 30);
40                 add(button_end);
41                 this.setVisible(false);
42         }
43         public void set_visible() {
44                 this.setVisible(true);
45         }
46         public void set_not_visible() {
47                 this.setVisible(false);
48         }
49
50 }
```

・Komadai_panel クラス

```
1  package swing;
2
3  import java.awt.BorderLayout;
4  import java.awt.Color;
5  import java.awt.Dimension;
6  import java.awt.GridBagConstraints;
7  import java.awt.GridBagLayout;
8  import java.awt.Image;
```

```
 9  import java.awt.Window;
10
11  import javax.swing.ImageIcon;
12  import javax.swing.JButton;
13  import javax.swing.JLabel;
14  import javax.swing.JPanel;
15  import javax.swing.border.LineBorder;
16
17  import board.Komadai;
18
19  public class Komadai_panel extends JPanel{
20          private GridBagLayout grid_bag_layout;
21          private JPanel[] panel_array;//0:歩
22          private Window him;
23
24          public Komadai_panel(Window her) {
25                  him = her;
26          }
27
28          public void change_color(int b) {
29                  JPanel panel = panel_array[b-1];
30                  if(panel != null) {
31                          panel.setOpaque(true);
32                  }
33          }
34          /*
35          public void change_color_back(int b) {
36                  JPanel panel = panel_array[b-1];
37                  if(panel != null) {
38                          panel.setOpaque(false);
39                  }
40          }
41          */
42          /**
43           * 駒台の一箇所の画像を変える
44           * @param b
45           * @param komadai
46           * @param action_listener
47           */
48          public void change_panel(int b,Komadai komadai,Action_listener action_listener) {
49                  GridBagLayout gbl = grid_bag_layout;
50                  GridBagConstraints gbc = new GridBagConstraints();
51                  int teban = komadai.get_teban();
52                  int koma_many = komadai.get_koma_number(b);
53                  JPanel old_panel = panel_array[b-1];
54                  if(old_panel != null) {
55                          old_panel.setVisible(false);
56                  }
57                  JPanel panel = new JPanel();
58                  panel.setOpaque(false);
59                  panel.setPreferredSize(new Dimension(45,50));
60                  panel.setBorder(new LineBorder(Color.black,1));
61                  panel.setLayout(new BorderLayout());
62                  int x = (b-1)%3;
63                  int y = (b-1)/3;
64                  if(teban==1) {
65                          gbc.gridx = x;
```

```
66                          gbc.gridy = y;
67                  }else {
68                          gbc.gridx = 2-x;
69                          gbc.gridy = 1-y;
70                  }
71                  gbl.setConstraints(panel, gbc);
72                  if(koma_many>0) {
73                          JLabel label = new JLabel(""+koma_many);
74                          ///駒の画像を表示させる。
75                          ImageIcon icon = Make_koma_image.make_koma_image(b, teban, him);
76                          Image image = icon.getImage().getScaledInstance(36, 40, Image.SCALE_DEFAULT);
77                          icon = new ImageIcon(image);
78                          JButton button = new JButton(icon);
79                          button.setContentAreaFilled(false);//背景透明化
80                          button.setBorderPainted(false);
81                          //actionlistener 加える
82                          button.addActionListener(action_listener);
83                          button.setActionCommand(""+teban+"0"+b);//駒台の駒の番号。盤の番号と同じ。先
    手の歩なら 11
84                          panel.add(button,BorderLayout.CENTER);
85                          panel.add(label,BorderLayout.NORTH);
86                          add(panel);
87                          panel_array[b-1] = panel;
88                  }else {
89                          //駒がない時
90                          JButton button = new JButton("");
91                          //actionlistener 加える
92                          button.addActionListener(action_listener);
93                          button.setActionCommand(""+teban+"0"+b);//駒台の駒の番号。盤の番号と同じ。先
    手の歩なら 11
94                          panel.add(button,BorderLayout.CENTER);
95                          add(panel);
96                          panel_array[b-1] = panel;
97                  }
98          }
99          public void set_komadai_panel(int p_w,int p_h) {
100                 setBackground(new Color(153,102,102));
101                 setBounds(p_w,p_h,200,120);//横、縦、幅、高さ
102                 GridBagLayout gbl = new GridBagLayout();
103                 grid_bag_layout = gbl;
104                 setLayout(gbl);
105                 make_komadai(gbl);
106                 panel_array = new JPanel[6];
107                 for(int i=0;i<6;i++) {
108                         panel_array[i] = null;
109                 }
110         }
111         private void make_komadai(GridBagLayout gbl) {
112                 GridBagConstraints gbc = new GridBagConstraints();
113                 for(int i=1;i<4;i++) {
114                         for(int j=1;j<3;j++) {
115                                 JPanel panel = new JPanel();
116                                 panel.setOpaque(false);
117                                 panel.setPreferredSize(new Dimension(45,50));
118                                 panel.setBorder(new LineBorder(Color.black,1));
119                                 gbc.gridx = i-1;
120                                 gbc.gridy = j-1;
```

```
121                                 gbl.setConstraints(panel, gbc);
122                                 add(panel);
123                         }
124                 }
125         }
126         public void make_komadai_panel(Komadai komadai,Action_listener action_listener) {
127                 int side = this.getX();
128                 int length = this.getY();
129                 removeAll();
130                 set_komadai_panel(side, length);
131                 GridBagLayout gbl = grid_bag_layout;
132                 GridBagConstraints gbc = new GridBagConstraints();
133                 int teban = komadai.get_teban();
134                 int koma_number = 1;
135                 if(teban==2) {
136                         koma_number = 6;
137                 }
138                 for(int j=1;j<3;j++) {
139                         for(int i=1;i<4;i++) {
140                                 int koma_many = komadai.get_koma_number(koma_number);
141                                 if(koma_many>0) {
142                                         JPanel panel = new JPanel();
143                                         panel.setOpaque(false);
144                                         panel.setPreferredSize(new Dimension(45,50));
145                                         panel.setBorder(new LineBorder(Color.black,1));
146                                         panel.setLayout(new BorderLayout());
147                                         gbc.gridx = i-1;
148                                         gbc.gridy = j-1;
149                                         gbl.setConstraints(panel, gbc);
150                                         JLabel label = new JLabel(""+koma_many);
151                                         ///駒の画像を表示させる。
152                                         ImageIcon icon = Make_koma_image.make_koma_image(koma_number, teban, him);
153                                         System.out.println(""+koma_number+","+i+","+j);
154         Image image = icon.getImage().getScaledInstance(36, 40, Image.SCALE_DEFAULT);
155                                         icon = new ImageIcon(image);
156                                         JButton button = new JButton(icon);
157                                         button.setContentAreaFilled(false);//背景透明化
158                                         button.setBorderPainted(false);
159                                         //actionlistener 加える
160                                         button.addActionListener(action_listener);
161                                         button.setActionCommand(""+teban+"0"+koma_number);//駒　台
    の駒の番号。盤の番号と同じ。先手の歩なら 11
162                                         panel.add(button,BorderLayout.CENTER);
163                                         panel.add(label,BorderLayout.NORTH);
164                                         add(panel);
165                                         panel_array[koma_number-1] = panel;
166                                 }else {
167                                         //駒台に駒がなくてもボタンは作る
168                                         JPanel panel = new JPanel();
169                                         panel.setOpaque(false);
170                                         panel.setPreferredSize(new Dimension(45,50));
171                                         panel.setBorder(new LineBorder(Color.black,1));
172                                         panel.setLayout(new BorderLayout());
173                                         gbc.gridx = i-1;
174                                         gbc.gridy = j-1;
175                                         gbl.setConstraints(panel, gbc);
176                                         ///駒の画像を表示させる。駒はないのでボタンだけ表示
```

```
177                                         JButton button = new JButton("");
178                                         //actionlistener 加える
179                                         button.addActionListener(action_listener);
180                                         button.setActionCommand(""+teban+"0"+koma_number);//駒　台
    の駒の番号。盤の番号と同じ。先手の歩なら 11
181                                         panel.add(button,BorderLayout.CENTER);
182                                         add(panel);
183                                         panel_array[koma_number-1] = panel;
184                                 }
185                                 if(teban==1) {
186                                         koma_number++;
187                                 }else {
188                                         koma_number--;
189                                 }
190                         }
191                 }
192         }
193 }
```

・Make_koma_Image クラス

```
1  package swing;
2
3  import java.awt.Window;
4
5  import javax.swing.ImageIcon;
6
7  /**
8   * 駒の数字から駒の Image を作る。
9   *
10  */
11 public class Make_koma_image {
12
13         public static ImageIcon make_koma_image(int koma_number,int teban, Window him) {
14                 ImageIcon image_icon = null;
15 //              ClassLoader cl = Make_koma_image.class.getClass().getClassLoader(); // to load image
16                 ClassLoader classloader = him.getClass().getClassLoader();
17 //              URL resUrl = classloader.getResource("hu.png"));
18                 switch(koma_number) {
19                 case 1:
20                         if(teban==1) {
21                                 image_icon = new ImageIcon(classloader.getResource("koma_image/hu.png"));
22 //                              image_icon = new ImageIcon("hu.png"));
23                         }else {
24                                 image_icon = new ImageIcon(classloader.getResource("koma_image/hu_t.png"));
25                         }
26                         break;
27                 case 2:
28                         if(teban==1) {
29                                 image_icon = new ImageIcon(classloader.getResource("koma_image/gin.png"));
30                         }else {
31                                 image_icon = new ImageIcon(classloader.getResource("koma_image/gin_t.png"));
32                         }
33                         break;
34                 case 3:
35                         if(teban==1) {
36                                 image_icon = new ImageIcon(classloader.getResource("koma_image/kin.png"));
37                         }else {
38                                 image_icon = new ImageIcon(classloader.getResource("koma_image/kin_t.png"));
```

```
39                          }
40                          break;
41                  case 4:
42                          if(teban==1) {
43                                  image_icon = new ImageIcon(classloader.getResource("koma_image/gyoku.png"));
44                          }else {
45                                  image_icon = new ImageIcon(classloader.getResource("koma_image/gyoku_t.png"));
46                          }
47                          break;
48                  case 5:
49                          if(teban==1) {
50                                  image_icon = new ImageIcon(classloader.getResource("koma_image/kaku.png"));
51                          }else {
52                                  image_icon = new ImageIcon(classloader.getResource("koma_image/kaku_t.png"));
53                          }
54                          break;
55                  case 6:
56                          if(teban==1) {
57                                  image_icon = new ImageIcon(classloader.getResource("koma_image/hisha.png"));
58                          }else {
59                                  image_icon = new ImageIcon(classloader.getResource("koma_image/hisha_t.png"));
60                          }
61                          break;
62                  case 11:
63                          if(teban==1) {
64                                  image_icon = new ImageIcon(classloader.getResource("koma_image/tokin.png"));
65                          }else {
66                                  image_icon = new ImageIcon(classloader.getResource("koma_image/tokin_t.png"));
67                          }
68                          break;
69                  case 12:
70                          if(teban==1) {
71                                  image_icon = new ImageIcon(classloader.getResource("koma_image/narigin.png"));
72                          }else {
73                                  image_icon = new ImageIcon(classloader.getResource("koma_image/narigin_t.png"));
74                          }
75                          break;
76                  case 15:
77                          if(teban==1) {
78                                  image_icon = new ImageIcon(classloader.getResource("koma_image/uma.png"));
79                          }else {
80                                  image_icon = new ImageIcon(classloader.getResource("koma_image/uma_t.png"));
81                          }
82                          break;
83                  case 16:
84                          if(teban==1) {
85                                  image_icon = new ImageIcon(classloader.getResource("koma_image/ryuu.png"));
86                          }else {
87                                  image_icon = new ImageIcon(classloader.getResource("koma_image/ryuu_t.png"));
88                          }
89                          break;
90                  }
91                  return image_icon;
92          }
93  }
```

・Menu_panel クラス

```
1   package swing;
```

```
2
3  import java.awt.Color;
4
5  import javax.swing.BoxLayout;
6  import javax.swing.JButton;
7  import javax.swing.JPanel;
8
9  public class Menu_panel extends JPanel{
10         private JButton button_new;//新規対局ボタン
11         private JButton button_start;//現在の局面から対局開始
12         private JButton button_touryou;
13         private JButton button_read_kyokumen;
14         private JButton button_read_kihu;
15         private JButton button_calc;
16         private JButton button_tumi;
17         private JButton button_save_kyokumen;
18         private JButton button_save_kihu;
19         private JButton button_matta;
20         private JButton button_free;
21         /**
22          * 対局中の状態にする。
23          */
24         public void set_gaming() {
25                 button_new.setVisible(false);
26                 button_start.setVisible(false);
27                 button_touryou.setVisible(true);
28                 button_read_kyokumen.setVisible(false);
29                 button_read_kihu.setVisible(false);
30                 button_calc.setVisible(true);//人間同士の時は隠したほうがいい?
31                 button_tumi.setVisible(true);
32                 button_save_kyokumen.setVisible(true);
33                 button_save_kihu.setVisible(true);
34                 button_matta.setVisible(true);
35                 button_free.setVisible(false);
36         }
37         public void set_not_gaming() {
38                 button_new.setVisible(true);
39                 button_start.setVisible(true);
40                 button_touryou.setVisible(false);
41                 button_read_kyokumen.setVisible(true);
42                 button_read_kihu.setVisible(true);
43                 button_calc.setVisible(true);
44                 button_tumi.setVisible(true);
45                 button_save_kyokumen.setVisible(true);
46                 button_save_kihu.setVisible(true);
47                 button_matta.setVisible(false);
48                 button_free.setVisible(true);
49         }
50         public void set_free_before() {
51                 button_new.setVisible(false);
52                 button_start.setVisible(false);
53                 button_touryou.setVisible(false);
54                 button_read_kyokumen.setVisible(false);
55                 button_read_kihu.setVisible(false);
56                 button_calc.setVisible(false);
57                 button_tumi.setVisible(false);
58                 button_save_kyokumen.setVisible(true);
```

```
59                  button_save_kihu.setVisible(false);
60                  button_matta.setVisible(false);
61                  button_free.setVisible(true);
62                  ///button_free.setText("終了");
63          }
64
65          public void set_menu_panel(Action_listener_menu action_listener_menu) {
66                  setBackground(Color.darkGray);
67                  setBounds(50,10,830,30);//横、縦、幅、高さ
68                  setLayout(new BoxLayout(this,BoxLayout.X_AXIS));
69                  button_new = new JButton("新規");//新規対局ボタン
70                  button_new.addActionListener(action_listener_menu);
71                  button_new.setActionCommand("新規対局");//
72                  add(button_new);
73                  button_start = new JButton("現始");//現在の局面から対局ボタン
74                  button_start.addActionListener(action_listener_menu);
75                  button_start.setActionCommand("現在の局面から対局");
76                  add(button_start);
77                  button_touryou = new JButton("投了");//投了ボタン
78                  button_touryou.addActionListener(action_listener_menu);
79                  button_touryou.setActionCommand("投了");
80                  add(button_touryou);
81                  button_read_kyokumen = new JButton("局読");//局面読み込みボタン
82                  button_read_kyokumen.addActionListener(action_listener_menu);
83                  button_read_kyokumen.setActionCommand("局面読み込み");
84                  add(button_read_kyokumen);
85                  button_read_kihu = new JButton("棋読");//棋譜読み込みボタン
86                  button_read_kihu.addActionListener(action_listener_menu);
87                  button_read_kihu.setActionCommand("棋譜読み込み");
88                  add(button_read_kihu);
89                  button_calc = new JButton("評価");//評価ボタン
90                  button_calc.addActionListener(action_listener_menu);
91                  button_calc.setActionCommand("局面評価");
92                  add(button_calc);
93                  button_tumi = new JButton("詰み");//詰みチェックボタン
94                  button_tumi.addActionListener(action_listener_menu);
95                  button_tumi.setActionCommand("詰みチェック");
96                  add(button_tumi);
97                  button_save_kyokumen = new JButton("局保");//局面保存チェックボタン
98                  button_save_kyokumen.addActionListener(action_listener_menu);
99                  button_save_kyokumen.setActionCommand("局面保存");
100                 add(button_save_kyokumen);
101                 button_save_kihu = new JButton("棋保");//棋譜保存チェックボタン
102                 button_save_kihu.addActionListener(action_listener_menu);
103                 button_save_kihu.setActionCommand("棋譜保存");
104                 add(button_save_kihu);
105                 button_matta = new JButton("待た");//待ったチェックボタン
106                 button_matta.addActionListener(action_listener_menu);
107                 button_matta.setActionCommand("待った");
108                 add(button_matta);
109                 button_free = new JButton("自由");//自由に駒を動かせるチェックボタン
110                 button_free.addActionListener(action_listener_menu);
111                 button_free.setActionCommand("自由");
112                 add(button_free);
113
114         }
115
```

```
116
117 }
```

・State_game クラス

```
  1 package swing;
  2
  3 import board.Board;
  4 import board.Move;
  5 import main.Kihu;
  6
  7 /**
  8  * 状態を保存するクラス。
  9  * 対局中とか、ある駒を選択しているとか
 10  */
 11
 12 public class State_game {
 13         //1:何もしていない,2:対局中手番は cp,3:対局中手番は人間で何も選択していない,4:対局中手番は人間で何かの駒を選択している。5:棋譜読み込んでいる。6:フリー自由に動かせる. 選択していない.7:選択されている
 14         private int state;
 15         private int before_place;
 16         private Kihu kihu;//棋譜を読み込んだ時に棋譜が保存される。
 17         private int count_tesuu;//棋譜を読み込んだ時になんて目のところにいるかを保存する。最初の局面にいる時は 0.
 18         private Board board;
 19         private Frame_s frame;
 20         public State_game(Board board,Frame_s frame) {
 21                 this.state = 1;
 22                 this.board = board;
 23                 this.frame = frame;
 24                 kihu = null;
 25                 count_tesuu = 0;
 26         }
 27         public void set_not_gaming() {
 28                 before_place = 0;
 29                 state = 1;
 30         }
 31         public void set_before_place(int before_place) {
 32                 //before_place を設定して手番が人間で駒を選択した状態にする。
 33                 this.before_place = before_place;
 34                 this.state = 4;
 35         }
 36         /**
 37          * 対局中で人間が手番の時 true
 38          * @return
 39          */
 40         public boolean check_gaming_player() {
 41                 if(state==3 || state == 4) {
 42                         return true;
 43                 }
 44                 return false;
 45         }
 46         public boolean check_chose_after_place() {
 47                 if(this.state == 4) {
 48                         return true;
 49                 }
 50                 return false;
 51         }
 52         /**
```

```
53          * 対局中かどうかを判断する。
54          * 対局をしていないなら true
55          * @return
56          */
57         public boolean check_not_gaming() {
58                 if(state==1 || state==5) {
59                         return true;
60                 }
61                 return false;
62         }
63         public boolean check_nothin() {
64                 if(state==1) {
65                         return true;
66                 }
67                 return false;
68         }
69         public boolean check_chose_before_koma() {
70                 if(state==3) {
71                         return true;
72                 }
73                 return false;
74         }
75         public boolean check_reading_kihu() {
76                 if(state==5) {
77                         return true;
78                 }
79                 return false;
80         }
81         public boolean check_free_before() {
82                 if(state==6) {
83                         return true;
84                 }
85                 return false;
86         }
87         public boolean check_free_after() {
88                 if(state==7) {
89                         return true;
90                 }
91                 return false;
92         }
93         public int get_before_place() {
94                 /*
95                 if(state != 4) {
96                         return -1;
97                 }
98                 */
99                 return before_place;
100        }
101        public int get_state() {
102                return state;
103        }
104        public void set_gaming(boolean player) {
105                //対局開始の状態にする。開始でない時も呼ばれる
106                if(player) {
107                        this.state = 2;//cp
108                        //次の一手を計算する。次の局面に進める。
109                        String teban = board.get_teban_s();
```

```
110                         if(board.go_next_board_from_swing()==0) {
111                                 //投了
112                                 frame.output_touryou(teban);
113                                 board.get_kihu().make_last_kyokumen();
114                                 set_reading_kihu(board.get_kihu().get_kihu_list().size());
115                                 frame.set_reading_kihu();
116                                 return;
117                         }
118                         Move move = board.get_before_move();
119                         frame.change_panel(move.get_after_place_a(), move.get_after_place_b(),
120                         board.get_kyokumen());
121                         frame.change_panel(move.get_before_place_a(), move.get_before_place_b(),
122                         board.get_kyokumen());
123                         move.output_move_kihu();
124                         if(move.get_get_koma() != 0) {
125                                 //駒を取っていた時
126                                 frame.change_panel((3-board.get_teban())*10, move.get_get_koma()%10,
127                                  board.get_kyokumen());
128                         }
129                         //cp同士でも反則があれば終了できる
130                         //駒を動かしたら反則がないか確認あればそこで対局終了
131                         if(board.check_foul()) {
132                                 System.out.println(board.get_teban_opposite_s()+"の反則負けです。！！");
133                                 frame.output_message(board.get_teban_opposite_s()+"の反則負けです。", "反則");
134                                 set_not_gaming();
135                                 frame.end_game();
136                         }
137                         int sennitite = board.check_sennitite();
138                         if(sennitite==1) {
139                                 //千日手
140                                 frame.output_message("千日手です。", "千日手");
141                                 set_not_gaming();
142                                 frame.end_game();
143                         }else if(sennitite==2) {
144                                 //連続王手の千日手。手番が王手している。
145                                 frame.output_message(board.get_teban_s()+"の反則負けです。", "反則:連続王手");
146                                 set_not_gaming();
147                                 frame.end_game();
148                         }else if(sennitite==3) {
149                                 //連続王手の千日手。手番が王手されている。
150                                 frame.output_message(board.get_teban_opposite_s()+"の反則負けです。", "反則:連続王手");
151                                 set_not_gaming();
152                                 frame.end_game();
153                         }
154                         //次もcpの時は待つために次へボタン表示
155                         if(board.get_teban_player()) {
156                                 //次へ行くか確認してやめることもできるようにする。
157                                 if(frame.check_next_or_exit()) {
158                                         //次へ行く
159                                         set_gaming(board.get_teban_player());
160                                 }else {
161                                         //そこで対局を終了する。
162                                         frame.output_message("対局を終了します。", "対局終了");
```

```
163                                     board.get_kihu().make_last_kyokumen();
164                                     set_reading_kihu(board.get_kihu().get_kihu_list().size());
165                                     frame.set_reading_kihu();
166                             }
167                     }else {
168                             //次が人の手番
169                             set_gaming(board.get_teban_player());
170                     }
171
172             }else {
173                     this.state = 3;//人間
174                     //入力を待つ
175             }
176             frame.set_next_te("手番: "+board.get_teban_s());
177     }
178     public void set_free_before() {
179             state = 6;
180             before_place = 0;
181     }
182     public void set_free_after() {
183             state = 7;
184     }
185     public void set_reading_kihu(int tesuu) {
186             state = 5;
187             before_place = 0;
188             this.kihu = board.get_kihu();
189             count_tesuu = tesuu;
190     }
191     public void set_tesuu_add() {
192             count_tesuu++;
193             if(count_tesuu>kihu.get_kihu_list().size()) {
194                     count_tesuu = kihu.get_kihu_list().size();
195                     System.out.println("error:count tesuu in state game class add ");
196             }
197     }
198     public void set_tesuu_decrease() {
199             count_tesuu--;
200             if(count_tesuu<0) {
201                     count_tesuu = 0;
202                     System.out.println("error:count tesuu in state game class decrease");
203             }
204     }
205     public void set_tesuu_first() {
206             count_tesuu = 0;
207     }
208     public void set_tesuu_end() {
209             count_tesuu = kihu.get_kihu_list().size();
210     }
211     public Kihu get_kihu() {
212             return kihu;
213     }
214     public int get_count_tesuu() {
215             return count_tesuu;
216     }
217     public Move get_move_next() {
218             if(count_tesuu >= kihu.get_kihu_list().size()) {
219                     return null;
```

```
220                 }
221                 return kihu.get_kihu_list().get(count_tesuu);
222         }
223         public Move get_move_back() {
224                 if(count_tesuu <= 0) {
225                         return null;
226                 }
227                 return kihu.get_kihu_list().get(count_tesuu-1);
228         }
229         public Board get_board() {
230                 return board;
231         }
232
233 }
```