卒業研究報告書

題目

放銃を避ける麻雀ゲームの AI 開発

指導教員

石水 隆　講師

報告者

14–1–037–0181

野田 竜希

近畿大学理工学部情報学科

平成 28 年 1 月 31 日提出

概要

本研究は零和有限不確定非完全情報ゲームにおける思考アルゴリズム研究の一貫として麻雀を題材にしたものである．麻雀は，ツモ運や配牌といった不確定要素により勝敗が大きく左右するため，完全な解析ができず必勝法は存在していない．麻雀の和了り方の一つに「ロンあがり」がある．ロンあがりは，自牌に自分以外の プレイヤーの捨て牌を加えると和了形となるときに，その牌を取って和了することである．相手の和了り牌を捨てること「放銃」といい，放銃すると得点を相手に支払わなければならない．つまり麻雀で勝つためには， 自分が和了ることだけでな，放銃しないようにもしなければいけない．そこで，本研究では放銃を避けることが勝率をあげることに繋がると仮定し，放銃を避けることを重視した麻雀アルゴリズムを作成する．本研究では放銃率を下げるためにいくつかのパラメータを作成し，それぞれに重みを与える．どのようなパラメータを作成し，どのくらいの重みを与えればよいのか，他の AI と対戦させデータをとることにより，最適な重みを与えたときに本当に放銃率が下がり，勝率が上がっているかを実証し検討する．

目次

1　序論

1.1　本研究の背景

麻雀は 2〜4 人で行う零和有限不確定非完全情報ゲームである．近年，チェスやオセロ，将棋や囲碁などの対戦をさせる，コンピュータで作成した思考アルゴリズムや人工知能の進歩が目まぐるしい．これらのボードゲームは零和有限確定完全情報ゲームに分類される．零和とは，全プレイヤーの点数の和が常に一定になることである．有限とは，双方のプレイヤーの手の組み合わせの総和が有限であることである．確定とは，運によりゲームの勝敗が左右されないことである．完全情報ゲームとは，お互いのプレイヤーがゲーム中終始，盤面での情報や，対戦相手の行動の選択を知ることができるゲームである．

これらに対し，麻雀や花札は零和有限不確定非完全情報ゲームに分類される．不確定とは，運により勝敗が左右されることであり，非完全情報ゲームとは相手の手札や，盤面の情報，相手の行動の選択が不明であることである．従ってこれらのゲームは相手の手札などが不明であることや，運により勝敗が大きく左右されることから完全解析が難しいとされている．最適解とされる行動を取っても必ず勝てるというわけではない．従って，自身の経験や，書籍を頼りに必要とする条件を与え，試行錯誤して勝率を上げるしかない．

麻雀には二種類の和了り方が存在する．一つはツモした牌を含めた自牌で和了る「ツモあがり」，もう一つは自牌と相手が捨てた牌を含めた牌で和了る「ロンあがり」がある．また，相手の和了り牌を捨てることを「放銃」という．前者は和了したプレイヤー以外のプレイヤーで得点を分け支払うのに対し，後者は和了り牌を捨てたプレイヤーが全得点を支払わなければならない．つまり，麻雀で勝つためには，自分が和了ることだけを考えるのではなく，相手の手を読み放銃しないことにも気を付けないといけない．

1.2　本研究の目的

前節で述べた通り，麻雀は放銃すると相手の得点を放銃したプレイヤーが全て支払わなければならない．そこで本研究で 放銃を避けるための思考を AI 化し，放銃率が低く勝率が高いアルゴリズムの開発を目指す．本研究では予め提供されている麻雀本体のプログラムを利用し，AI インタフェイスを用いて AI の開発を図る．また [1] の AI，3 つと対戦させ放銃率，勝率を数値化し客観的に放銃しない AI か検討する．

1.3　本報告書の構成

本報告書の構成を以下に述べる．2 章では麻雀ゲームについての簡単な概要．3 章では AI 作成までの準備．4 章では作成した AI と [1] の AI との対戦結果を考察する．5 章ではこれまでに得られた結果から導き出された結果から今後の課題について述べる，

2　麻雀について

本章では麻雀のルールについて以下に述べる．

2.1　麻雀ゲームの概要

麻雀は 3〜4 人で行うゲームである．本研究では 4 人で行うものとして述べる．麻雀は 34 種の牌をそれぞれ 4 枚づつの全 136 枚の牌を使用する．

まず最初にプレイヤーの中から親を決める．そして，時計回りランダムに 13 枚の牌（以下，配牌とする）が配られ，残りの牌は山として置かれる．その後親から順番に，1 枚山から牌を引き（以下，ツモとする），ツモした牌を含めた 14 枚の手牌から 1 枚捨てる行為を反時計回りに繰り返す．これらの中で他プレイヤーより早く決められた役を目指す．

1 章で述べた通りツモした後の 14 枚の手配で和了形が完成した場合「ツモ」を宣言する．ここでの「ツモ」は「ツモあがり」の「ツモ」であり，前述のツモとは異なる．また，自分以外のプレイヤーの手番中で，手番プレイヤーが捨てた牌 1 枚と自分の手牌 13 枚を合わせた 14 枚で和了形ができる場合「ロン」を宣言する．このとき手番プレイヤーが和了り牌を捨てることを「放銃」という．

和了形に含まれる役の組み合わせにより得点が決まっておりそれに沿って点数のやり取りを行う．ツモあがりした場合，和了したプレイヤーに対して他のプレイヤーが分担して得点を支払う．このとき親は基本的に 1.5 倍の得点を貰えるが，代わりに支払うべき点数は 2 倍になる．一方，ロンで和了したときは，和了したプレイヤーに対して，放銃したプレイヤーが全ての得点を支払う．配牌から和了までの一連の流れを 1 局という．局が終われば，全ての牌をシャッフルし直し，次の局を始める．もし山から引ける牌が無くなっても誰も和了できなかった場合はその時点でその局は終わり，同様に全ての牌をシャッフルし直し，次局を始める．親が和了した場合は和了したプレイヤーは親を継続し，子が和了した場合は反時計回りに親を移動する．流局した場合は，親が後一枚の牌和了できる場合（以下，テンパイとする），親が継続する．そうでない場合（以下，ノーテンとする）は親を移動する．また，ノーテンの場合はペナルティとして点数を支払わなければならない．

通常，麻雀は半荘と呼ばれる 8 局 (全員が 2 回親になるまで，ゲーム過程により増減する) を 1 試合とする．半荘終了後，最終的に各プレイヤーの持ち点の過多により勝敗を決める．従って，プレイヤーの目指すべき事は如何に高い点数を得るか，如何に相手へ点数を渡さないかに尽きる．

2.2　フリテンについて

ここでは麻雀のルールの一つ「フリテン」について説明する．フリテンとは，他プレイヤーからロンあがりができない状態であることを指す．以下のうちどれかに当てはまればフリテン状態になる．

- アガリ牌を自分が捨てている
- 自分が牌を捨ててから次のツモまでにアガリ牌が捨てられている
- アガリ牌をリーチ後に見逃した

1 つ目の「アガリ牌を自分で捨てている」はアガリ牌が複数あっても，どれか一つでも捨ててしまっているとフリテン状態になる．

3　研究内容

3.1　AI 開発の準備

本研究で作成する AI は，フリーソフトウェアである「まうじゃん for Java」[1] プログラム (以下，ベースプログラムとする) をベースとしている．本研究で作成する麻雀 AI は，ベースプログラムで提供されているインタフェイス (麻雀をする上で最低限の状況や牌を読み取るクラス) を取得し，得られた情報を元に複合的に条件指定をすることにより AI として機能させる．

3.2　戦略的クラスの概要

本研究では麻雀のツモ運や場の流れ（一時的に偏った確率事象を取り上げる）と行った非合理的な要素を鉄笛的に排除し，場を見て得られる全ての情報を集約した合理的な評価関数の導入を前提としている．麻雀における思考は大きく分けて 2 つある．自分の手番での行動（リーチするか・カン・流局するか・捨て牌を考える）と相手の手番での行動（ポン・チー・カン・ロンをする．以下副露とする）本 AI では行動の指針として評価関数及び手牌の評価値を取り入れている．同じ牌を扱うにしても状況により牌の評価は常に変動している．基本的には牌効率（和了まで最も早く且つ確率の高い手順）を重視している．本研究では相手の手番での行動はロンにのみ限る．その理由は，捨てることのできる牌を多く残すことにより，安全牌が増えるからである．

3.2.1　自分の手番での戦略

捨て牌の戦略

自分の手番で山場から牌をツモったとき，和了していなければ 14 枚の手牌の中から 1 枚を捨てなければならない．従って，麻雀 AI は自分の手番時には，場の状況から得られる情報からどの牌を捨てるかを導き出さなければならない．どの牌を捨てるかを決めるための戦略を以下に挙げる．

- 面子や対子が出来ている牌の組み合わせを残す
  これは現在の手牌かの中で，面子または対子となっている牌以外から捨て牌を選ぶ戦略である．この戦略を取ると和了率が上がる．

- リーチしているプレイヤーを警戒する
  これは，リーチしているプレイヤーを警戒する戦略である．リーチしているプレイヤーを警戒せずに捨て牌すると，放銃率が上がり，同時に勝率が下がる可能性がある．

- 局の指定した巡目以降は全プレイヤーを警戒する
  これは，指定した巡目以降から全プレイヤーを警戒し始める戦略である．麻雀のルールの一つに副露しているとリーチすることが出来ないというルールがある．また，役ができていればリーチをしなくても和了することができることから，指定した巡目以降は，他プレイヤーがテンパイしている可能性を考慮し，警戒する．

- 偏った捨て牌をしているプレイヤーは警戒する
  これは，麻雀の役の一つである「混一色」や「清一色」などを警戒するための戦略である．混一色とは，萬子，索子，筒子のうちどれか一種類の牌と，字牌のみで和了形が形成されていた場合の役である．清一色は，萬子，索子，筒子のうちどれか一種類のみで和了形が形成されていた場合の役である．どちらも強力な役であるため，警戒をすると放銃率が下がり，勝率も上がる．他プレイヤーの捨て牌の中でどれか一種類極端に捨てられていない場合に警戒する．

- スジを警戒する
  これは相手の和了牌を読む一つのテクニック「スジ」を用いて放銃率を下げる戦略である．スジとは，2.2 節で説明した，フリテンになるルールのうちの「アガリ牌を捨てている」というルールを用いた他プレイヤーの和了牌を読むテクニックである．以下に図 1，図 2 を用いて例を挙げる．

図 1　他プレイヤーが持っている牌

図 2　他プレイヤーが待っている牌

  他プレイヤーがテンパイであって図 1 の面子を持っているとする．するとこのプレイヤーの待ちは図 2 の二種類になる．ここで，このプレイヤーが四萬を捨てていたとするとフリテン状態になり，一萬を捨ててもこのプレイヤーはロンすることができない．つまり，四萬が捨てられていることによって，その 2 つ上下の牌は安全牌である確率が高い．今回のような四萬を捨てていると一萬と七萬が安全牌である確率が高いということになる．このような相手の和了牌の読み方を「スジ」と呼ぶ．図 1 のような形で和了牌を待つことをリャンメン待ちという．リーチ時におけるリャンメン待ちである確率は 65.1% [2] と高い確率であるので，この読みはかなり有効であると言える．

手牌の評価

前節のそれぞれの戦略から導き出される捨て牌は 1 つに定まらない場合が多々ある．そこで本研究で作成する麻雀 AI では，手牌を評価するアルゴリズムを導入する．これにより手牌の価値が最も高くなるように捨てるべき 1 つの牌を選出する．以下，手牌を評価するアルゴリズムについて述べる．

$$E_{just} = \sum_i p_i E_i$$

E_{just} : 現在の手牌の評価値
i : 次のツモ牌
p_i : 次に牌 i が来る確率
E_i : 次に牌 i が来た時の手牌の評価値

手牌の評価値は，手牌中の完成面子数と未完成面子数及び未完成面子に対する待ち牌が来る確率を求める．次にツモる可能性のある牌の種類およびその確率を，手牌およびプレイヤーが目視出来る場の情報 (捨て牌・

ポン・チー・カンされた牌およびそれに付随している牌・ドラ表示牌) から求める．次に仮に手牌からある牌を一つ捨てた場合，次にツモる可能性のある牌の種類とその確率から，各牌をツモった場合の手牌の評価値の期待値をそれぞれの牌を捨てた場合の評価値から計算し，最も評価値を高くする牌を捨てる．

捨て牌の評価値

捨て牌は，手牌 1 枚 1 枚を評価し各牌の危険度を求める．危険度は設定した条件に当てはまった場合設定した値を足していく．本研究で放銃を避けるために判断する条件として A(他プレイヤーがテンパイしてい るかもしれないと警戒し始める巡目) ，B(A で指定した巡目以降に係る係数)，C(他プレイヤーのリーチに対する危険度)，D(相手が同じ種類の牌ばかりを集めていそうな時の危険度)，E(スジに対する危険度) とする．8〜10 巡目で平均的にテンパイするので，A を 9 巡目に設定し，他の条件の効果を調べる．[2] 以下，捨て牌の評価をするアルゴリズムについて説明する．

$$E_a = \sum_{k=1}^{3} \omega_{anpai_k} \omega_{B_k} \omega_{C_k} \left(\omega_{D_k} + \omega_{E_k} + \omega_{dora} + \omega_{jihai} \right)$$

E_a : 捨て牌の評価値
k : プレイヤー番号
ω_{anpai_k} : プレイヤー k が捨てている牌なら 0 それ以外なら 1
ω_{B_k} : A で指定した巡目以降の危険度，プレイヤー k がリーチしていたら，1
ω_{C_k} : 他プレイヤーがリーチしている時の危険度
ω_{D_k} : プレイヤー k が混一色，清一色を狙っていそうな時の危険度
ω_{E_k} : プレイヤー k のスジに対する危険度
ω_{dora} : ドラに対する危険度
ω_{jihai} : 字牌に対する危険度

捨て牌 a に対する評価値 E_a は，以下の操作を各プレイヤー $k(k = 1, 2, 3)$ に対して行い，合計することで求められる．

1. k が a を捨てていれば安全牌なので危険度 0 とする．捨てていない場合は危険度を上げる．
2. a がドラであれば危険度を上げる．
3. a が字牌の場合，a が 1 枚も捨てられていなければ危険度を上げる．
4. k が a のスジ牌を捨てていれば危険度を下げる
5. k が a と同じ種類の牌を捨てていなければ混一色，清一色狙いの可能性があるので危険度を上げる
6. k がリーチをかけていれば今まで加算した危険度に指定した係数を掛ける．k がリーチをしておらず，かつ A で指定した巡目以降なら危険度に指定した係数を掛ける．

3.3 麻雀 AI プログラム

本節では，本研究で作成した麻雀 AI プログラムについて述べる．付録に本研究で作成した麻雀 AI プログラムのソースを示す．

3.3.1 クラス AIP

クラス AIP は本研究で作成した麻雀 AI プログラムの中心部分である．以下のクラス AIP の主なメソッドについて述べる．

- int onSutehai
  ツモしてからの行動を返すメソッドである．リーチをしていれば自動的にツモ牌を捨てる．和了形ができていればツモあがりをする．テンパイ且つ，一度も副露していなければ後述する calcReachOrNot メソッドを参照し，点数や残りのツモ回数を考えてリーチするかどうかを判断する．
- int calc_sutehai
  calc_sutehai() メソッドは手牌から捨て牌を選ぶメソッドである．ツモした後の手牌から 1 枚ずつ，その牌を捨てた後の手牌の評価と，その牌を捨てる牌自体に対する評価の合計が最も高いものが捨て牌となる．
- int eval_tehai
  eval_tehai() メソッドは手牌を評価するメソッドである．後述する eval_connection() メソッドで手牌内の牌の組み合わせに対する評価を行い，eval_hai メソッドで牌 1 つ 1 つに対し評価を行う．これらの合計が手牌に対する評価値となる．
- int eval_hai
  eval_hai() メソッドは牌 1 つ 1 つを評価するメソッドである．本研究では，数牌，ドラ，赤ドラであれば評価値をプラスとする．
- int eval_tehai_connection，int eval_tehai_connection_3，void eval_tehai_connection_3_sub
  これらは手牌の評価関数を計算するメソッドである．手牌の牌の組み合わせや点数を考慮して捨て牌を選出する．
- int eval_sutehai
  捨て牌の危険度を評価するメソッドである．3.2.1 節で説明した通りに評価し，牌の危険度を設定する．
- int calcNaki
  calcNaki() メソッドは鳴くかどうかを判断する．ロンができる場合，親である場合や，和了ることにより順位が上がる場合にはロンあがりする．鳴ける場合には後述する eval_tehai_in_naki メソッドで評価し，鳴くかどうかを判断する．
- int eval_tehai_in_naki
  eval_tehai_in_naki メソッドは，鳴いた牌，その後に捨てる牌，捨てた後の手牌の評価の合計を評価値とする．

4　結果・考察

本研究では，1 章で述べたとおり，[1] の AI 3 つと作成した AI で対戦を行った．

4.1　対戦結果

表に 300 局を対局したデータを示す．表 1 中の勝率は 1 位，2 位であった確率の合計である．放銃率は，放銃した確率を示している．

表 1　対戦結果

重み					放銃率	勝率
A	B	C	D	E		
5	2	3	3	5	9.0%	19.2%
9	2	3	3	5	9.2%	16.0%
15	2	3	3	5	12.3%	7.6%
9	2	1	3	5	16.3%	11.5%
9	1	3	3	5	9.3%	8.7%
9	2	3	3	0	12.0%	8.7%
9	2	3	0	5	9.3%	5.7%

4.2　考察

3.2.1 節で示した，放銃を避ける条件について，有効であったか 1 より考察する．

A(他プレイヤーがテンパイしているかもしれないと警戒し始める巡目)

警戒し始める巡目が早ければ早い方が放銃率が低くなり，勝率が高くなった．放銃率を下げることで，勝率も上がり，AI として強くなることが示された．

B(A で指定した巡目以降に係る係数)

A で指定した巡目以降にリーチしていない他プレイヤーを警戒するときとしないときでは，放銃率にはあまり違いがなかった．勝率には大きな違いが出たが，原因はわからなかった．

C(他プレイヤーのリーチに対する危険度)

リーチしているプレイヤーに対する警戒をしないときは放銃率がとても上がることが示された．

D(相手が同じ種類の牌ばかりを集めていそうな時の危険度)

混一色，清一色を警戒しない場合には，放銃率にはあまり違いが見られなかったが，勝率に大きな差が見られた．混一色や清一色は強い役となるため，放銃した際に多い点数を支払わなければならないためである．

E(スジに対する危険度)

スジに対する警戒をしなければ，放銃率が上がることが示された．3.2.1 節で述べたとおり，リャンメン待ちでテンパイする確率がとても高いため，効果があったと考えられる．

5　結論と今後の課題

本研究では放銃を避ける麻雀の AI 開発を行った．AI 同士を対戦させた結果，放銃率の低い AI を作成することができた．しかし麻雀は不確定非完全情報ゲームであり，1.1 節よりランダム要素が絡むので結果的にどの局面において必ずしも正しい判断が出来ているとは言いきれない．

今後の課題として，今の放銃率を保ったままの勝率の向上が挙げられる．そのためには，今回指定した条件を局の最初から動作させるのではなく，クセや戦略を読み取り今回設定した条件の重みを AI 自身で設定したり，様々な戦略に切り替えることができることなどが望まれる．

謝辞

本研究を行うにあたり，ご指導いただきました石水隆講師には，1年半という短い期間ではありましたが，大変お世話になりました．日頃からの研究に対するアドバイスや議論，適切で丁寧な意見や励ましの言葉をいただきましたので，この場を借りて感謝を申し上げます．

参考文献

[1] 石畑恭平. コンピュータ麻雀のアルゴリズム. 工学社, 2007.
[2] とつげき東北. 科学する麻雀. 洋泉社, 2009.

付録 A　麻雀 AI のソースプログラム

```
1  import jp.gr.java_conf.ishihata.mj_ai.*;
2
3  public class Hoju extends MJ_AI {
4          private MIPIface iface;
5
6          int te_cnt[] = new int[34]; // 各牌の数
7
8          private boolean my_anpai[] = new boolean[34]; // 自分の安全牌
9          private int max_mentsu_suu; // 刻子，順子，雀頭の最大数
10         private boolean tehai_machi[] = new boolean[34]; // この手牌の待ち
11         private int nokori_hais[] = new int[34]; // 各牌の残り数
12
13         int i_oras_kyoku; // オーラスの局
14
15         private int visible_hais_add[] = null; // 残り牌計算時に、すでに見えている
   牌として処理すべき牌のはい番号の配列
16
17         private final static int SCORE_TOITSU = 4; // 順子の評価点
18         private final static int SCORE_KOTSU = 8; // 刻子の評価点
19         private final static int SCORE_SHUNTSU_MACHI = 2; // 順子待ちの評価点
20         private final static int SCORE_MENTSU = 50; // 面子の評価点
21         private final static int SCORE_ATAMA = 45; // 雀頭の評価点
22         private final static int SCORE_MACHI = 1; // 待ち牌の残り数に対する評価点
23         private final static int SCORE_KAZUHAI = 1; // 数牌に対する評価点
24         private final static int SCORE_DORA = 1; // ドラに対する評価点
25         private final static int SCORE_ANZEN = 1; // 安全度に対する係数
26         private final static int SCORE_REACH_KIKEN = 3; // リーチの危険度を表す
   係数
27         private final static int SCORE_SOMETE = 0; // 染め手に対する危険度を表す評
   価点
28         private final static int SCORE_DAMATEN = 2; // 終盤での危険度
29         private final static int JUNME = 9;
30         private final static int SCORE_SUJI = 5;
31
32         public boolean initialize(MIPIface i) {
```

```
33                  this.iface = i;
34                  return true;
35          }
36
37          public String getName() {
38                  return "AI";
39          }
40
41          /**
42           * ツモしてからの行動
43           */
44          public int onSutehai(MJITehaiReader te, MJIHaiReader tsumohai) {
45                  if (iface.getAgariScore() > 0)
46                          return MJPIR_TSUMO; // ツモ
47                  if (iface.isKKHaiable())
48                          return MJPIR_NAGASHI; // 九種九牌時に流す
49
50                  // このメソッドに渡される手配オブジェクトは読み取り専用クラスなのでte
51                  // 操作可能なクラスのオブジェクトへコピーしておくMJITehai
52                  // (メソッドの中で利用するcalc_sutehai)
53                  MJITehai tehai = new MJITehai(te);
54
55                  // 捨てる牌のインデックスを計算する
56                  int sutehai_index = calc_sutehai(tehai, tsumohai);
57
58                  // リーチするか判断する
59                  if (calcReachOrNot(tehai, tsumohai, sutehai_index))
60                          return MJPIR_REACH | sutehai_index;
61                  return MJPIR_SUTEHAI | sutehai_index;
62          }
63
64          /**
65           * 手牌から捨て牌を選ぶ
66           *
67           * @param tehai
68           * 手牌
69           * @param tsumohai
```

```
70              * ツモってきた牌，ツモ牌がない場合は                null
71              * @return 捨てる牌のインデックス．ツモ牌を捨てる場合はを返す．13
72              */
73             private int calc_sutehai(MJITehai tehai, MJIHaiReader tsumohai) {
74                     int selected_hai = 0; // 捨てる牌のインデックス．とりあえず左端の牌を
    捨てるようにセットする．
75                     int score_max = -1; // 最大評価値
76
77                     // 順手牌の配列を取得する
78                     MJIHaiReader tehai_hai[] = tehai.getTehai();
79
80                     // もしツモ牌がある場合は，そのツモ牌を捨てた場合の評価を出しておく
81                     if (tsumohai != null) {
82                             // 捨てた後の手牌に対する評価を計算する""
83                             int tehai_score = eval_tehai(tehai_hai, tsumohai);
84
85                             // 捨てる牌自体に対する評価を計算する""
86                             int sutehai_score = eval_sutehai(tsumohai);
87
88                             // 総合評価
89                             // とりあえずこの値を最大評価値としておき，他の捨て牌の場合の評価
    値と比較する．
90                             score_max = tehai_score + sutehai_score;
91                             selected_hai = 13;
92                     }
93
94                     // 持っている牌を一つずつ，捨てた時の評価点を計算して，
95                     // もっとも評価点が高くなる捨て牌をとる
96                     for (int i = 0; i < tehai_hai.length; i++) {
97                             // 捨てる牌
98                             MJIHaiReader sutehai = tehai_hai[i];
99
100                            // もし一つ前の牌と同じ牌ならスキップ
101                            if (i > 0)
102                                    if (sutehai.equals(tehai_hai[i - 1]))
103                                            continue;
104                            // もしこの牌が捨てられない牌ならスキップ
```

```
105                         if (!iface.isHaiThrowable(sutehai.getHaiNo()))
106                                 continue;
107
108                         // 捨てる牌をツモ牌と交換する
109                         // もしツモ牌がない場合になるnull
110                         tehai_hai[i] = tsumohai;
111
112                         // 捨てた後の手牌に対する評価を計算する""
113                         int tehai_score = eval_tehai(tehai_hai, sutehai);
114
115                         // 捨てる牌自体に対する評価を計算する""
116                         int sutehai_score = eval_sutehai(sutehai);
117
118                         // 総合評価
119                         int score = tehai_score + sutehai_score;
120
121                         // これまでの最大値より高い評価値なら捨て牌候補をこの牌に変更
122                         if (score > score_max) {
123                                 selected_hai = i;
124                                 score_max = score;
125                         }
126
127                         // 手牌情報を元の状態に戻す
128                         tehai_hai[i] = sutehai;
129                 }
130
131                 return selected_hai;
132         }
133
134         /**
135          * 手牌を評価するメソッド
136          */
137         private int eval_tehai(MJIHaiReader tehai_hai[], MJIHaiReader sutehai) {
138                 // 牌の組み合わせに対する評価を算出""
139                 int ret = eval_tehai_connection(tehai_hai, sutehai);
140
141                 // ひとつひとつの牌に対する評価を加算していく""
```

```
142                 for (int i = 0; i < tehai_hai.length; i++) {
143                         ret += eval_hai(tehai_hai[i]);
144                 }
145                 return ret;
146         }
147
148         /**
149          * 牌ひとつひとつを評価するメソッド
150          *
151          * @param hai
152          * 評価する牌
153          * @return ret 評価点
154          */
155         private int eval_hai(MJIHaiReader hai) {
156                 int ret = 0; // 戻り値
157
158                 // 数牌なら評価点プラス
159                 if (hai.getHaiNo() < 27)
160                         ret += SCORE_KAZUHAI;
161
162                 // ドラであれば評価点プラス
163                 int[] doras = iface.getDora();
164                 for (int i = 0; i < doras.length; i++)
165                         if (hai.getHaiNo() == doras[i])
166                                 ret += SCORE_DORA;
167
168                 // 赤ドラであれば評価点プラス
169                 if (hai.hasAttribute(MJIHaiReader.ATTR_RED))
170                         ret += SCORE_DORA;
171
172                 return ret;
173         }
174
175         /**
176          * 手牌配列から配列を構築するメソッドte_cnt
177          *
178          * @param tehai_hai
```

```
179          * 手牌配列
180          */
181         private void setTeCnt(MJIHaiReader tehai_hai[]) {
182                 for (int i = 0; i < 34; i++)
183                         te_cnt[i] = 0;
184
185                 for (int i = 0; i < tehai_hai.length; i++) {
186                         te_cnt[tehai_hai[i].getHaiNo()]++;
187                 }
188         }
189
190         private int eval_tehai_connection(MJIHaiReader tehai_hai[],
191                         MJIHaiReader sutehai) {
192                 // 手牌配列から配列を構築するte_cnt
193                 setTeCnt(tehai_hai);
194
195                 // 捨てようとしている牌番号
196                 int sutehai_hai = -1; // 何も捨てない場合は-1
197                 if (sutehai != null)
198                         sutehai_hai = sutehai.getHaiNo();
199
200                 // return eval_tehai_connection_1(sutehai_hai);
201                 // return eval_tehai_connection_2(sutehai_hai);
202                 return eval_tehai_connection_3(sutehai_hai);
203         }
204
205         private int eval_tehai_connection_3(int sutehai) {
206
207                 // 自分の安全牌をセットする
208                 // また、この手牌の待ちを初期化する
209                 for (int i = 0; i < 34; i++) {
210                         my_anpai[i] = iface.isHaiAnpai(0, i);
211                         tehai_machi[i] = false;
212                 }
213                 setNokoriHais();
214                 // 捨てようとしている牌も安全牌にして，その残りの数も減らす
215                 if (sutehai >= 0) {
```

```
216                         my_anpai[sutehai] = true;
217                         nokori_hais[sutehai]--;
218                 }
219
220                 // 刻子，順子，雀頭を探し，その数がもっとも多い組み合わせを見つけて
221                 // その数をにセットする．また,その時の待ちをにセットする．max_mentsu_suutehai_machi
222                 max_mentsu_suu = 0;
223
224                 eval_tehai_connection_3_sub(false, 0);
225
226                 // 刻子，順子，雀頭の評価点
227                 int score_mentsu = max_mentsu_suu * 50;
228
229                 // 待ち牌に対する評価
230                 int score_machi = 0;
231                 for (int i = 0; i < 34; i++) {
232                         if (tehai_machi[i])
233                                 score_machi += nokori_hais[i];
234                 }
235
236                 // 最終的な手牌の評価点
237                 int score = score_mentsu + score_machi;
238
239                 return score;
240         }
241
242         private void eval_tehai_connection_3_sub(boolean atama_flag, int mentsu_suu) {
243                 boolean nothing_flag = true;
244                 for (int p = 0; p < 34; p++) {
245                         if (te_cnt[p] == 0)
246                                 continue;
247                         int c = te_cnt[p];
248
249                         if (c >= 2 && !atama_flag) {
250                                 te_cnt[p] -= 2;
251                                 eval_tehai_connection_3_sub(true, mentsu_suu + 1);
252                                 nothing_flag = false;
```

```
253                             te_cnt[p] += 2;
254                     }
255                     if (c >= 3) {
256                             te_cnt[p] -= 3;
257                             eval_tehai_connection_3_sub(atama_flag, mentsu_suu + 1);
258                             nothing_flag = false;
259                             te_cnt[p] += 3;
260                     }
261
262                     if (p < 27) {
263                             int kazu = (p % 9) + 1;
264                             if (kazu < 8) {
265                                     if (te_cnt[p + 1] > 0 && te_cnt[p + 2] > 0) {
266                                             te_cnt[p]--;
267                                             te_cnt[p + 1]--;
268                                             te_cnt[p + 2]--;
269                                             eval_tehai_connection_3_sub(atama_flag,
270                                             nothing_flag = false;
271                                             te_cnt[p]++;
272                                             te_cnt[p + 1]++;
273                                             te_cnt[p + 2]++;
274                                     }
275                             }
276                     }
277             }
278             if (!nothing_flag)
279                     return;
280
281             // 刻子、順子、雀頭が存在しなかった場合
282             // ここまでに数えた面子数が最大面子数より少なかったらここまで
283             if (mentsu_suu < max_mentsu_suu)
284                     return;
285
286             // ここまでに数えた面子数が最大面子数より大きかったら
287             // 最大面子数を更新して待ち牌情報をクリアする
288             if (mentsu_suu > max_mentsu_suu) {
289                     max_mentsu_suu = mentsu_suu;
```

```
290                     for (int i = 0; i < 34; i++)
291                             tehai_machi[i] = false;
292             }
293
294             // 残りの牌で待ちを確認する
295             for (int p = 0; p < 34; p++) {
296                     if (te_cnt[p] == 0)
297                             continue;
298                     // 対子
299                     if (te_cnt[p] >= 2 && !my_anpai[p])
300                             tehai_machi[p] = true;
301
302                     // まだ雀頭がなかったら、雀頭も待つ
303                     else if (!atama_flag && my_anpai[p])
304                             tehai_machi[p] = true;
305
306                     // ペンチャン、カンチャン、二面待ち
307                     if (p < 27) {
308                             int kazu = (p % 9) + 1;
309                             // カンチャン
310                             if (kazu < 8) {
311                                     if (te_cnt[p + 2] > 0) {
312                                             if (!my_anpai[p + 1]) {
313                                                     tehai_machi[p + 1] = true;
314                                             }
315                                     }
316                             }
317                             // ペンチャン、二面待ち
318                             if (kazu < 9) {
319                                     if (te_cnt[p + 1] > 0) {
320                                             if (kazu > 1) {
321                                                     if (!my_anpai[p - 1]) {
322                                                             tehai_machi[p - 1] = tru
323                                                     }
324                                             }
325                                             if (kazu < 8) {
326                                                     if (!my_anpai[p + 2]) {
```

```
327                                                                     tehai_machi[p + 2] = tru
328                                                                 }
329                                                             }
330                                                         }
331
332                                                 }
333                                             }
334                                         }
335                                 }
336
337                                 /**
338                                  * 捨て牌の危険度を評価するメソッド
339                                  *
340                                  * @param hai
341                                  * 捨て牌
342                                  * @return から危険度を引いた数を評価値としている50
343                                  */
344                                 private int eval_sutehai(MJIHaiReader hai) {
345                                         int ds = 0; // 危険度
346                                         int tsumohai_remain = iface.getHaiRemain(); // 残りツモ牌の数
347
348                                         for (int i = 1; i < 4; i++) {
349                                                 // 安牌なら危険度0
350                                                 if (iface.isHaiAnpai(i, hai.getHaiNo()))
351                                                         continue;
352
353                                                 int plds = 0; // このプレイヤーに対する危険度
354                                                 int hai_no = hai.getHaiNo(); // 捨て牌の牌番号
355
356                                                 // 安牌でなければ危険度アップ1
357                                                 plds++;
358
359                                                 // スジのチェック
360                                                 if (hai_no < 27) {
361                                                         int kazu = (hai_no % 9) + 1;
362                                                         boolean fl = true;
363                                                         if (kazu > 3)
```

```
364                                     if (!iface.isHaiAnpai(i, hai_no - 3))
365                                             fl = false;
366                                     if (fl)
367                                             if (kazu < 7)
368                                                     if (!iface.isHaiAnpai(i, hai_no + 3))
369                                                             fl = false;
370                                     // スジでなければ危険度アップ1
371                                     if (!fl)
372                                             plds += SCORE_SUJI;
373                             } else {
374                                     // 字牌の場合、初牌なら危険度アップ1
375                                     if (iface.getVisibleHais(hai_no) == 0)
376                                             plds++;
377                             }
378
379                             /*
380                              * 染め手なら危険度アップツモ数がまだまだ残っていたら危険度はその
    まま
381                              */
382                             MJIKawahaiReader[] kawa = iface.getKawa(i);
383                             double p1 = 0.0;
384                             int p2 = 0, hai_shu = 0;
385                             if (!(kawa.length == 0)) {
386                                     // マンズ
387                                     if (hai_no >= 0 && hai_no < 9) {
388                                             for (int j = 0; j < kawa.length; j++) {
389                                                     if (kawa[j].getHai().getHaiNo() >= 0
390                                                                     && kawa[j].getHai().getH
391                                                             hai_shu++;
392                                                     p2 = hai_shu / kawa.length;
393                                                     p1 = (double) p2;
394                                             }
395                                     }
396                                     // ピンズ
397                                     else if (hai_no >= 9 && hai_no < 18) {
398                                             for (int j = 0; j < kawa.length; j++) {
399                                                     if (kawa[j].getHai().getHaiNo() >= 9
```

```
400                                                     && kawa[j].getHai().getH
401                                                 hai_shu++;
402                                         }
403                                 }
404                                 // ソウズ
405                                 else if (hai_no >= 18 && hai_no < 27) {
406                                         for (int j = 0; j < kawa.length; j++) {
407                                                 if (kawa[j].getHai().getHaiNo() >= 18
408                                                         && kawa[j].getHai().getH
409                                                 hai_shu++;
410                                         }
411                                 }
412
413                                 if (tsumohai_remain < 70 - JUNME * 4) // 巡
    目以降JUNME
414                                         if (p1 <= 0.1)
415                                                 plds += SCORE_SOMETE;
416                         }
417
418                         // リーチだったら危険度アップ
419                         if (iface.isPlayerReached(i))
420                                 plds *= SCORE_REACH_KIKEN;
421
422                         // リーチしてなくても巡目以降なら危険度アップJUNME
423                         else if (tsumohai_remain < 70 - JUNME * 4)
424                                 plds *= SCORE_DAMATEN;
425
426                         ds += plds;
427                 }
428                 return 20 - ds;
429         }
430
431         /**
432          * ゲームが始まるときに行う処理
433          */
434         public void inStartGame() {
435                 // オーラスの局をセットしておく
```

```
436                 if (iface.getRule(MIPIface.MJRL_NANNYU) == 0)
437                         i_oras_kyoku = 3;
438                 else if (iface.getRule(MIPIface.MJRL_SHANYU) == 0)
439                         i_oras_kyoku = 7;
440                 else
441                         i_oras_kyoku = 15;
442         }
443
444         /**
445          * リーチをかけるかどうかを判断するメソッド
446          *
447          * @param te
448          * 手牌
449          * @param tsumohai
450          * ツモした牌
451          * @param sutehai_x
452          * 捨て牌
453          * @return リーチをするかどうか
454          */
455         public boolean calcReachOrNot(MJITehai te, MJIHaiReader tsumohai,
456                         int sutehai_x) {
457                 int tsumohai_remain = iface.getHaiRemain(); // 残りツモ牌の数
458
459                 // 次のツモ牌がなければリーチできない
460                 if (tsumohai_remain < 4)
461                         return false;
462
463                 // 鳴いてないか？
464                 if (te.getMinkans().length + te.getMinshuns().length
465                                 + te.getMinkos().length > 0)
466                         return false;
467
468                 // 配列をセットしておくte_cnt
469                 MJIHaiReader[] tehai_hai = te.getTehai();
470                 setTeCnt(tehai_hai);
471
472                 // 捨てようとしている牌番号
```

```
473                 int sutehai;
474
475                 // 捨て牌した後の手牌を用意する
476                 if (sutehai_x < tehai_hai.length) {
477                         sutehai = tehai_hai[sutehai_x].getHaiNo();
478                         te.removeHaiFromTehai(sutehai_x);
479                         te.addHaiToTehai(tsumohai);
480                 } else {
481                         sutehai = tsumohai.getHaiNo();
482                 }
483
484                 // テンパってるか？
485                 boolean[] machi = new boolean[34];
486                 boolean b_tenpai = iface.getMachi(te, machi);
487
488                 // テンパイでないならリーチしない
489                 if (!b_tenpai)
490                         return false;
491
492                 // 待ち牌がいくつ残っているか数える
493                 // またあがったときの点数を計算する
494                 int machihai_remain = 0;
495                 int agari_score = 0;
496                 for (int i = 0; i < 34; i++) {
497                         if (machi[i]) {
498                                 // もしフリテンだったらリーチしない
499                                 if (iface.isHaiAnpai(0, i) || i == sutehai)
500                                         return false;
501
502                                 // すでに場に出ている牌を数える
503                                 int disp_num = iface.getVisibleHais(i) + te_cnt[i];
504                                 if (tsumohai != null)
505                                         if (tsumohai.getHaiNo() == i)
506                                                 disp_num++;
507
508                                 // もしこの牌が全て出てしまっているなら次の牌へ
509                                 if (disp_num >= 4)
```

```
510                                     continue;
511
512                             machihai_remain += 4 - disp_num;
513
514                             // このときのあがり点
515                             int sc = iface.getAgariScore(te, i);
516                             if (sc == 0 || sc < agari_score)
517                                     agari_score = sc;
518                     }
519             }
520
521             // 待ち牌が一つも残っていないならリーチしない
522             if (machihai_remain == 0)
523                     return false;
524
525             // もしリーチをかけなくてもそこそこに点があるならリーチしない
526             if (agari_score >= 7700)
527                     return false;
528             // オーラスかどうかを判断
529             if (iface.getKyoku() == i_oras_kyoku) {
530                     // オーラスの場合
531                     // トップとの点差を計算する
532                     int top_score = iface.getScore(1);
533                     for (int i = 2; i < 4; i++) {
534                             int sc = iface.getScore(2);
535                             if (sc > top_score)
536                                     top_score = sc;
537                     }
538                     int my_score = iface.getScore(0); // 自分の点数
539                     int score_sa = top_score - my_score; // トップとの差
540
541                     // もし自分がトップならリーチはかけない
542                     // また、リーチをかけなくても逆転可能な点差ならリーチはかけない
543                     if (score_sa < agari_score)
544                             return false;
545
546                     // リーチをかける
```

```
547                     return true;
548             }
549
550             // リーチをかけた場合のあがり点
551             int reached_agari_score = agari_score * 2;
552
553             // もし役なしなら、ドラの数を数える
554             if (agari_score == 0) {
555                     int dora[] = iface.getDora();
556                     int doras = 0; // ドラの数
557                     for (int i = 0; i < dora.length; i++) {
558                             doras += te_cnt[dora[i]];
559                             if (tsumohai != null)
560                                     if (tsumohai.getHaiNo() == dora[i])
561                                             doras++;
562                             if (sutehai == dora[i])
563                                     doras--;
564                     }
565                     // 赤ドラも探す
566                     for (int i = 0; i < tehai_hai.length; i++) {
567                             if (tehai_hai[i].hasAttribute(MJIHaiReader.ATTR_RED))
568                                     doras++;
569                     }
570                     // アンカンについてもドラを探す
571                     MJIHaiReader ankan_hai[][] = te.getAnkans();
572                     for (int i = 0; i < ankan_hai.length; i++) {
573                             // この牌がドラか？
574                             int hai = ankan_hai[i][0].getHaiNo();
575                             for (int j = 0; j < dora.length; j++) {
576                                     if (hai == dora[i])
577                                             doras += 4;
578                             }
579                             // 赤ドラを探す
580                             for (int j = 0; j < 4; j++) {
581                                     if (ankan_hai[i][j].hasAttribute(MJIHaiReader.AT
582                                             doras++;
583                             }
```

```
584                         }
585
586                         // リーチした場合の上がり点を計算する
587                         int han_suu = doras + 3;
588                         reached_agari_score = 30 << han_suu; // 府計算での基本
    点30
589                         if (reached_agari_score >= 2000) {
590                                 // 満貫以上の計算
591                                 if (han_suu < 8)
592                                         reached_agari_score = 2000; // 満貫
593                                 else if (han_suu < 10)
594                                         reached_agari_score = 3000; // 跳満
595                                 else if (han_suu < 13)
596                                         reached_agari_score = 4000; // 倍満
597                                 else if (han_suu < 15)
598                                         reached_agari_score = 6000; // 三 倍
    満
599                                 else
600                                         reached_agari_score = 8000; // 数 え
    役満
601                         }
602                         if (iface.getCha() == 0)
603                                 reached_agari_score *= 6;
604                         else
605                                 reached_agari_score *= 4;
606                         reached_agari_score += 90;
607                 }
608
609                 // 待ち牌の出現率を計算する
610                 // まず「全ての待ち牌が現れない確率」を計算する
611                 double p = 1.0;
612                 for (int i = 0; i < machihai_remain; i++) {
613                         p *= (66 - i) / (66 - i + tsumohai_remain);
614                 }
615                 // 待ち牌のうちの最低一つが現れる確率
616                 p = 1.0 - p;
617                 // リーチをかけることで他家に警戒されるためか率は下がる
618                 p /= 2.0;
```

```
619
620                 // リーチをかけることで得られるメリットを評価する
621                 int reach_score = (int) (((reached_agari_score - agari_score) / 100) * p
622                 return reach_score >= 30;
623         }
624
625         /**
626          * 何かが起きた時に呼び出されるメソッド
627          *
628          * @param action
629          * 行動
630          * @param player_no
631          * 行動を起こしたプレイヤー
632          * @param target_no
633          * 対象となったプレイヤー
634          * @param hai
635          * 牌オブジェクト
636          * @return 行動
637          */
638         public int onAction(int action, int player_no, int target_no, MJIHai hai) {
639                 switch (action) {
640                 case MJPIR_REACH: // リーチ
641                 case MJPIR_SUTEHAI: // 捨て牌
642                         if (player_no != 0) {
643                                 // 自分以外の捨て牌だったら、鳴くか否かを判断
644                                 return calcNaki(hai, player_no);
645                         }
646                         return 0; // なにもしない
647
648                 case MJPIR_ANKAN: // アンカン
649                 case MJPIR_MINKAN: // ミンカン
650                         if (player_no != 0) {
651                                 // 自分以外のカンだったら、ロンするかを判断
652                                 int sel_ron = calcRon(target_no);
653                                 if (sel_ron == 2)
654                                         return MJPIR_RON; // ロン
655                         }
```

```
656                         return 0; // なにもしない
657
658                 }
659                 return 0; // なにもしない
660         }
661
662         /**
663          * 鳴くかどうか判断するメソッド
664          *
665          * @param hai
666          * 対象となってる牌
667          * @param player_no
668          * 対象となってる牌を捨てたプレイヤー
669          * @return ポンするかチーするか
670          */
671         private int calcNaki(MJIHaiReader hai, int player_no) {
672                 // ロンするか？
673                 int sel_ron = calcRon(player_no);
674                 if (sel_ron == 2)
675                         return MJPIR_RON; // ロン
676                 if (sel_ron == 1)
677                         return 0; // テンパイしている
678
679                 MJITehaiReader tehai = iface.getTehai();
680                 MJIHaiReader tehai_hai[] = tehai.getTehai();
681                 setTeCnt(tehai_hai);
682                 int hai_no = hai.getHaiNo(); // 場に出た牌の牌番号
683
684                 // すでに鳴いているか？
685                 boolean menzen = tehai.getMinkans().length + tehai.getMinkos().length
686                                 + tehai.getMinshuns().length == 0; // 面前の
    場合true
687
688                 // まず面前の場合の処理
689                 if (menzen) {
690                         // 手に同一牌が枚あって、かつ字牌であること2
691                         if (hai_no >= 27 && te_cnt[hai_no] == 2) {
```

```
692                             // 字牌であってもオタ風ではだめ
693                             if (hai_no >= 31 || hai_no - 27 == iface.getCha()
694                                             || hai_no - 27 == iface.getKyoku() / 4)
695                                     // ドラの場合はポンする
696                                     int[] doras = iface.getDora();
697                                     for (int i = doras.length - 1; i >= 0; i--) {
698                                             if (doras[i] == hai_no)
699                                                     return MJPIR_PON;
700                                     }
701                                     // 自分が十分に勝っている場合はポンする
702                                     int my_score = iface.getScore(0);
703                                     int max_sc_sa = 0;
704                                     for (int i = 1; i < 4; i++) {
705                                             int sc_sa = my_score - iface.getScore(i
706                                             if (sc_sa > max_sc_sa)
707                                                     max_sc_sa = sc_sa;
708                                     }
709                                     if (max_sc_sa > 8000)
710                                             return MJPIR_PON;
711                             }
712                     }
713                     // 残りのツモ牌が少ない場合は形式テンパイを目指すための下の処理へ
714                     // そうでない場合はここまで
715                     if (iface.getHaiRemain() > 12)
716                             return 0;
717             }
718
719             // 現在の手牌の評価点を計算する
720             int max_score = eval_tehai(tehai_hai, null);
721
722             int ret = 0;
723
724             // すでに副露してる場合は、価値のある牌だけ鳴く
725             // ポンの判定
726             if (te_cnt[hai_no] >= 2) {
727                     // この評価点に、鳴いた部分の評価点を加える
728                     int sc = eval_tehai_in_naki(tehai, hai, hai_no, hai_no);
```

```
729                     if (sc > max_score) {
730                             max_score = sc;
731                             ret = MJPIR_PON;
732                     }
733             }
734
735             // チーの判定
736             // 上家以外の人が捨てた牌はチーできない
737             // また字牌はチーできない
738             if (player_no == 3 && hai_no < 27) {
739                     int kazu = (hai_no % 9) + 1; // 数牌の数の値
740
741                     // 左端をチーする場合
742                     if (kazu < 8)
743                             if (te_cnt[hai_no + 1] > 0 && te_cnt[hai_no + 2] > 0) {
744                                     int sc = eval_tehai_in_naki(tehai, hai, hai_no +
745                                                     hai_no + 2);
746                                     if (sc > max_score) {
747                                             max_score = sc;
748                                             ret = MJPIR_CHII1;
749                                     }
750                             }
751
752                     // 右端をチーする場合
753                     if (kazu > 2)
754                             if (te_cnt[hai_no - 1] > 0 && te_cnt[hai_no - 2] > 0) {
755                                     int sc = eval_tehai_in_naki(tehai, hai, hai_no -
756                                                     hai_no - 2);
757                                     if (sc > max_score) {
758                                             max_score = sc;
759                                             ret = MJPIR_CHII2;
760                                     }
761                             }
762
763                     // 真ん中をチーする場合
764                     if (kazu > 1 && kazu < 9)
765                             if (te_cnt[hai_no - 1] > 0 && te_cnt[hai_no + 1] > 0) {
```

```
766                                     int sc = eval_tehai_in_naki(tehai, hai, hai_no -
767                                             hai_no + 1);
768                                     if (sc > max_score) {
769                                             max_score = sc;
770                                             ret = MJPIR_CHII3;
771                                     }
772                             }
773                     }
774                     return ret;
775             }
776
777             /**
778              * から、とのはい番号の牌を取り除いた上でtehainaki_hai_no1naki_hai_no2
779              * 捨てたはいを決定し、その捨て牌を捨てた後の評価点を返す
780              *
781              * @param tehai
782              * 手牌オブジェクト
783              * @param target_hai
784              * 牌番号
785              * @param naki_hai_no1
786              * 牌番号
787              * @param naki_hai_no2
788              * 牌番号
789              * @return
790              */
791             private int eval_tehai_in_naki(MJITehaiReader tehai,
792                             MJIHaiReader target_hai, int naki_hai_no1, int naki_hai_no2) {
793                     MJITehai temp_tehai = new MJITehai(tehai);
794
795                     // 鳴く対象の牌枚を探して手牌オブジェクトから取り除く2
796                     MJIHaiReader removed_hai1 = removeHaiFromTehaiForNaki(temp_tehai,
797                                     naki_hai_no1);
798                     MJIHaiReader removed_hai2 = removeHaiFromTehaiForNaki(temp_tehai,
799                                     naki_hai_no2);
800
801                     // 鳴いた後の捨て牌を決める
802                     int sutehai_index = calc_sutehai(temp_tehai, null);
```

```
803                 MJIHaiReader temp_tehai_hai[] = temp_tehai.getTehai();
804                 MJIHaiReader sutehai = temp_tehai_hai[sutehai_index]; // 捨
    てる牌
805                 // その捨て牌を捨てた後の手牌オブジェクトを生成
806                 temp_tehai.removeHaiFromTehai(sutehai_index);
807                 temp_tehai_hai = temp_tehai.getTehai();
808                 // を呼ぶ前に、鳴く対象の牌枚を「すでに見えている牌」にしておくeval_tehai2
809                 visible_hais_add = new int[2];
810                 visible_hais_add[0] = naki_hai_no1;
811                 visible_hais_add[1] = naki_hai_no2;
812                 // 手牌評価
813                 int temp_tehai_score = eval_tehai(temp_tehai_hai, sutehai);
814                 visible_hais_add = null;
815                 // この評価点に、鳴いた部分の評価点を加える
816                 int sc = temp_tehai_score + eval_hai(removed_hai1)
817                                 + eval_hai(removed_hai2) + eval_hai(target_hai) + SCORE_
818
819                 return sc;
820         }
821
822         private MJIHaiReader removeHaiFromTehaiForNaki(MJITehai tehai, int hai_no) {
823                 MJIHaiReader[] tehai_hai = tehai.getTehai();
824
825                 int index = 0;
826                 for (int i = 0; i < tehai_hai.length; i++) {
827                         if (tehai_hai[i].getHaiNo() == hai_no) {
828                                 // もし赤牌ならこれで決定
829                                 index = i;
830                                 if (tehai_hai[i].hasAttribute(MJIHaiReader.ATTR_RED))
831                                         break;
832                         }
833                 }
834                 MJIHaiReader hai = tehai_hai[index];
835                 tehai.removeHaiFromTehai(index);
836                 return hai;
837         }
838
```

```
839         /**
840          * 各牌の残り数をセットする
841          */
842         private void setNokoriHais() {
843                 for (int i = 0; i < 34; i++) {
844                         nokori_hais[i] = 4 - iface.getVisibleHais(i) - te_cnt[i];
845                 }
846                 if (visible_hais_add != null) {
847                         for (int i = 0; i < visible_hais_add.length; i++) {
848                                 nokori_hais[visible_hais_add[i]]--;
849                         }
850                 }
851         }
852
853         /**
854          * ロンするか否かを判断するメソッド
855          *
856          * @return ロンしない0: テンパイだがロンしない1: ロンする2:
857          */
858         private int calcRon(int player_no) {
859                 // テンパイか否か、テンパイの場合はその待ちを取得する
860                 boolean machi[] = new boolean[34];
861                 boolean b_tenpai = iface.getMachi(machi);
862
863                 // テンパイの場合は上がれるかチェックする
864                 if (b_tenpai) {
865                         int agari_score = iface.getAgariScore(); // 純粋な上
    がり点
866                         if (agari_score > 0) {
867                                 // フリテンチェック
868                                 for (int i = 0; i < 34; i++) {
869                                         if (machi[i])
870                                                 if (iface.isHaiAnpai(0, i))
871                                                         return 1;
872                                 }
873                                 // 親なら上がる
874                                 if (iface.getCha() == 0)
```

```
875                     return 2;
876
877             /*
878              * 子の場合は、自分の点数と一つ上の順位の人の点数を比較しその点差と残りの局数から判断する
879              */
880             int tensa = 0; // 自分と一つ上の順位の人の点差
881             int rank = 0; // 自分の順位
882             int my_score = iface.getScore(0); // 自分の点差
883             for (int i = 1; i < 4; i++) {
884                     int sa = iface.getScore(i) - my_score;
885                     if (sa > 0) {
886                             rank++;
887                             if (tensa == 0 || sa < tensa)
888                                     tensa = sa;
889                     }
890             }
891
892             // もし現在トップなら上がる
893             if (rank == 0)
894                     return 2;
895
896             agari_score += iface.getHonba() * 300; // 本場数にかかる点を含める
897             int get_score = agari_score + iface.getReachBou() * 1000 // 供託リーチ棒を含める
898
899             // オーラスの場合とそうでない場合で分ける
900             if (iface.getKyoku() == i_oras_kyoku) {
901                     // オーラスの場合は順位が上がる場合だけあがる
902                     // あがった後の点数分布を計算する
903                     int agari_go_score[] = new int[4];
904                     for (int i = 0; i < 4; i++) {
905                             agari_go_score[i] = iface.getScore(i);
906                     }
907                     agari_go_score[0] += get_score;
908                     agari_go_score[player_no] -= agari_score;
```

```
909                                             // あがった後の順位を確認する
910                                             int agari_go_rank = 0;
911                                             for (int i = 1; i < 4; i++) {
912                                                     if (agari_go_score[i] > agari_go_score[0
913                                                             agari_go_rank++;
914                                             }
915                                             if (agari_go_rank < rank)
916                                                     return 2;
917                                     } else {
918                                             // オーラスでない場合
919                                             int border = tensa / (i_oras_kyoku - iface.getKy
920                                             if (get_score >= border)
921                                                     return 2;
922                                     }
923                                     return 1;
924                             }
925                             // テンパイだけどあがれない場合はなにもしない
926                             return 1;
927                     }
928                     return 0;
929             }
930     }
```