卒業研究報告書

題目

素早く和了を目指す麻雀ゲーム AI の開発

指導教員

石水 隆　講師

報告者

17–1–037–0216

中野 圭悟

近畿大学理工学部情報学科

令和 3 年 2 月 1 日提出

概要

近年，様々なゲームで AI 開発が盛んであり，ゲームによっては人間が勝つことができない強さを持つ AI が開発されている．しかし，麻雀は「不完全情報ゲーム」であり，たとえ効率が最大の行動をしても運要素により勝てるとは限らず，強い AI を作ることは難しいとされる．麻雀は役を作ってあがり，点数を稼ぐゲームであり，役の 1 つに「リーチ」がある．これは，対戦相手に自分がテンパイであることを明確に示すものであり，対戦相手は警戒せねばならず，降りも選択肢となることが多い．つまり，早い局面の段階で「リーチ」を宣言することで，自分があがることができるだけでなく，対戦相手に振り込む危険性が下がるため，勝率が上がると考えられる．そこで，本研究では素早くテンパイを目指すことを重視した麻雀アルゴリズムを作成し，他の AI と対戦させ，麻雀において素早くテンパイすることの強さ，リーチによる対戦相手へのプレッシャーを勝率や和了率により検証する．また，麻雀における基本動作である「鳴き」と「降り」をそれぞれ組み込み，「鳴き」によるさらに素早いテンパイの強さ，「降り」により放銃を減らすことで勝率をさらに上げることも検証する．

目次

1 序論

1.1 本研究の背景

近年，将棋やチェス，囲碁などのゲーム AI の進化が目まぐるしく，人間が AI に勝つことが難しくなったゲームが増えてきた．これらのゲームは運要素や相手の動きが鮮明であることから確定完全情報ゲームと分類され，完全解析が可能とされている．ただし，莫大な局面，盤面を計算する必要があるため，今日のコンピュータでは完全な計算ができないため，実際に完全解析されたゲームはごく一部である．一方，麻雀は零和有限不確定非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総和が常に同じであることである．有限とは，プレイヤーが取れる選択肢が有限であることである．不確定とは，シャッフルやサイコロ等を用い，運要素が絡むことである．非完全ゲームとは，ゲーム上において，相手や山などにより情報を全て得られないことである．つまり，効率が最も良い行動をしても運により勝敗が左右され，対戦相手の手牌や思考が読み取れないため，麻雀は完全解析どころか，強い AI を作ることすら難しいとされてきた．

1.2 既存の麻雀 AI

前節で述べた通り，麻雀はその不確定性，非完全情報であることから，将棋や囲碁と比べるといい手を選択することが難しく，AI の作成が難しい．また，近年ではディープラーニングによるゲーム AI の研究が注目されているが，麻雀は学習させるデータに乏しく，将棋や囲碁のようにプロの棋譜を学習させることも難しい．そのため，結果を残す麻雀 AI が登場したのは比較的最近のことである．

自分の手牌と相手の捨て牌から各牌の残り枚数を算出し，手を進める手法はシンプルかつ和了率が高く，多数研究されてきたが，相手の手を読むことが難しく，降りることができないため放銃率は高くなってしまい，強くするのは困難であった．

近年では，トッププレイヤーに迫る麻雀 AI が開発されている．2015 年に東京大学が開発した「爆打」，2018 年にドワンゴが開発した「NAGA25」，2019 年にマイクロソフトが開発した「Suphx」の 3 つが強い成績を残している．特に「Suphx」は他の 2 つの AI がゲーム「天鳳」の安定段位が 6.5 段であるにもかかわらず，8.7 段という好成績を出している．これはトッププレイヤー (7.5 段) より強い結果であり [6]，凄まじい強さであることがわかる．

1.3 麻雀の戦略

麻雀の役の 1 つに「リーチ」がある．リーチは，手牌の入れ替えができなくなる代わりに，あがれた際に裏ドラをめくり，得点を加算する可能性ができるものである．また，対戦相手に明確にテンパイであることを示すため，ツモあがりよりも振り込みによるロンあがりの方が痛手である麻雀では，リーチされると降りることが多い．つまり，対戦相手より早くテンパイすることで，自分があがることができ，運が良ければ裏ドラにより点数を稼ぐことができると同時に，対戦相手に降りさせることで自分が振り込む危険性が下がるため，麻雀で勝ちやすくなると考えることができる．

また，麻雀には「ツモあがり」，「ロンあがり」の 2 つのあがりがあり，ツモはあがったプレイヤーに対して他プレイヤー全員があがり点を分担して払うが，ロンでは振り込んだプレイヤーがあがり点全額を上がったプレイヤーに払わなければならず，振り込みを避けることも麻雀に勝つことにおいて重要である．振り込みを避

けるには，相手の捨てた牌そのものを捨てることが 1 番安全である．これは麻雀のルールである「フリテン」を利用しており，フリテンとは，自分が 1 度捨てた牌があがり牌に含まれる時，ロンあがりできないというルールである．また，麻雀ではあがりを待つ時にリャンメン待ち (並んだ数牌 2 枚の左右の数牌 2 枚を待つこと) であることが多く [3]，これとフリテンを利用し「スジ」という考え方でさらに振り込みを避けることができる．詳しくは，3.2.3 節で述べる．

麻雀は，あがりを目指すだけでは振り込みが増え，振り込みを避けるだけでは点数が稼げないため，あがりを目指す戦略と振り込みを避ける戦略のバランスがとても重要である．一般的には，誰かがリーチを宣言，もしくは鳴きにより手牌が少なくなった状況，自分がゲーム終盤まで手があまり進まない状況で降りを始めることが多く，それまでは自分の手を進めることに尽力する．この判断はプレイヤーや順位状況などによって大きく変わり，これが麻雀の面白い点である．

1 位を目指す際には，「親」によりあがり点が 1.5 倍となっている時にあがることが重要である．さらに親はあがるか流局時にテンパイしていることで継続となるため，攻めるべきと言える．逆に，「子」である時は親には特に振り込まないよう注意することで，大きな減点を抑えることができ，結果として最下位になりにくくなる．なお，最下位で勝負の終盤となってしまった時は，点が高い手を作り一発逆転を目指すこともできるが，危険度が高いことや現実性が薄いことから安い手で 1 つでも上の順位を目指す方がよい．

1.4 既知の類似研究

佐藤らは，4 種類の異なる戦略を取る麻雀 AI に対してその有効性を検証している [5]．佐藤らは，有効牌が最も多くなるように打牌選択をする AI1，1 つ先のツモ後の打牌で有効牌が最も多くなるなるように打牌する AI2，AI1 にテンパイの際に有効牌の枚数に和了点で重みをつける AI3，AI1 にドラや数牌を優先して残す AI4 を用意し，それぞれ人間プレイヤと対戦させている．どの AI も和了率は上位プレイヤーと同程度であるが，放銃率，最終レートで大きな差が出ている．また，各 AI のレートの差が少量であり，明らかな強さの差は見られなかった．

1.5 本研究の目的

本研究では，素早くあがりを目指すため，有効牌の種類と枚数から打牌選択をするとともに，佐藤らの研究では考慮されていなかった鳴きと降りをそれぞれ実装し，勝率をさらに高くさせるアルゴリズムの開発を目指す．

1.6 本報告書の構成

本報告書の構成を以下に述べる．2 章では麻雀についてのルールと用語，3 章では作成した AI の戦略やプログラムの概要，4 章では対戦内容とその結果・考察，5 章では結果から導き出される結論と今後の課題について述べる．

2　麻雀について

2.1　麻雀のルール

麻雀は 3〜4 人で行うゲームであり，マンズ，ピンズ，ソウズという 3 種類の 1〜9 の数牌と東西南北・白發中の 7 種類の字牌を用いる．全ての牌は 4 枚ずつ使用する．プレイヤーは手牌として 13 枚の牌が配られ，順番に山から牌を 1 枚引き，手牌と入れ替えもしくは引いた牌をそのまま捨てる．これを繰り返し，4 つの面子と 1 つの雀頭があと 1 枚で出来上がる状況と役を作り，自分があがり牌を引きツモあがり，もしくは相手が自分のあがり牌を捨てることによるロンあがりをして点数を稼ぐ．面子とは順子と刻子と槓子の 3 つから構成され，順子とは同種類の数牌で連続した 3 つの数字の牌のまとまり，刻子は同じ牌 3 つからなるまとまりのこと，槓子は同じ牌 4 つからなるまとまりのことを言う．雀頭は同じ牌 2 枚からなるまとまりである．例外として，七対子，国士無双の 2 役は 4 つの面子と 1 つの雀頭からは成らない．

プレイヤーは東家，南家，西家，北家にそれぞれ振り分けられ，東家は親，それ以外は子となる．親はあがった場合，子のあがり点数と比べて 1.5 倍の点数が獲得でき，自分が親のまま次の局へ移行する．子があがるか，誰もあがれず局が流れた場合，プレイヤーの方角を反時計回りに 1 つ回転させ，次の局へ移行する．

あがりにはツモあがりとロンあがりがあり，ツモあがりの場合は全員が均等にあがったプレイヤーに点数を払う．ただし，親が子にツモあがりされた場合は，あがり点の半分を親，もう半分を他プレイヤーで払う．ロンあがりの場合は振り込んだプレイヤーが全てのあがり点を払う必要がある．誰も上がれず，局が流れた場合はその時点でテンパイできていないプレイヤーはテンパイしているプレイヤーに点数を払う必要がある（ノーテン罰符）．

麻雀は最初の場風は東から始まり，1 回転したら反時計回りに場風が回転する．ゲームとして，東風まで対戦する東風戦，南風まで対戦する半荘戦の 2 つが一般的である．最終的な持ち点で順位がつけられる．

2.2　麻雀の用語

本節では本研究で用いる用語の説明を行う．

- 和了
  ホーラ．あがりのことである．ツモあがりであるかロンあがりであるかは問わない．
- テンパイ
  あと 1 枚であがれる状況のこと．手牌が進みきった状態である．
- シャンテン数
  テンパイまでに必要な有効牌の数のこと．0 になった時，テンパイとなる．
- 門前
  メンゼン．手牌において 1 回も鳴いていない状態のこと．
- 鳴き
  相手の捨て牌時に宣言することで，相手に見せた状態でその捨て牌を組み込むことができる．ポン・チー・カンの 3 種類あり．ポンは刻子，チーは順子が相手の捨て牌で完成するときに宣言できる．ただし，チーは左方のプレイヤーからしか宣言できない．カンは本研究では使用しない．
- リーチ

麻雀の役の 1 つであり，門前でテンパイした際に宣言できる．宣言した場合，手牌の入れ替えができなくなるが，あがれた際には裏ドラをめくることができる．

- ドラ・裏ドラ
  ドラは麻雀におけるあがれた際のボーナスの存在である．役ではないが，役と同じく点数が加算される．局開始時に 1 枚にめくられる．裏ドラはリーチを宣言してあがった場合にさらにボーナスでめくるドラのことである．
- 役牌
  字牌には役つきのものがあり，役牌と呼ばれる．方角を示す 4 つの牌では，自分の方角と現在の場風の方角に役がつく．白發中の 3 牌は状況によらず常に役となる．
- フリテン
  テンパイの際，あがり牌をすでに自分が捨て牌としている場合もしくはあがりを見逃した際に発生するペナルティ．相手からのロンあがりができなくなる．また，自分のあがり牌が複数ある時に 1 種類でもフリテンが発生すると全てのあがり牌でロンあがりできなくなる．

3 研究内容

本章では，作成した AI とその戦略，プログラムについて述べる．

3.1 AI 作成の準備

本研究では，[1] の AI インターフェイスを用いた思考ルーチンと麻雀ゲームのプログラムを元に，Java を用いて麻雀 AI を作成する．このプログラムで提供されているインターフェイスを取得し，取得した情報から取るべき戦略を条件指定し，AI として機能させる．インターフェイスには，麻雀の盤面，牌，点数を読み取るクラスが搭載されている．

3.2 戦略

本節では，本研究で作成した麻雀 AI の戦略について述べる．本研究で作成する麻雀 AI は安い手であっても早く和了することを目指す．点数で負けているときも高い手を狙わないので，結果として最下位を避ける AI となる．

3.2.1 手牌を進めるための戦略

本研究のベースとなる戦略である．手牌を効率よく進めるためには，面子数を減らさず，シャンテン数を減らすため，待つ有効牌の数を多くするように捨て牌を選択すればよい．そこで，自分の手番で山から牌をツモし，14 枚の手牌から捨てる牌を選ぶ時に以下の順番で 1 枚ずつ牌を評価し，1 番評価値の高い牌を捨てる．

1. 手配から仮に 14 枚のうち 1 枚捨てたとする．
2. 次の手番で 1 枚牌をツモる．この時の面子数，雀頭の有無を計算する．麻雀の牌は 34 種類あるため，これを 1 種類ずつ計算する．なお，すでに場に同じ牌が 4 枚確認できる場合はその牌は計算しない．
3. 34 種類のツモの中で 1 番面子数と雀頭を合わせた数が多い 1 種類を抽出し，評価関数で計算し，現在の手配からこの 1 枚を仮に捨てた時の評価点数とする．この時，面子数と雀頭を合わせた数が同じ牌が

あれば，待ち牌の残り数の合計が多い方を抽出する．以下に評価関数を記述する．

$$f = (\text{面子数と雀頭を合わせた数} * 50) + \text{待ち牌の合計数} \tag{1}$$

そのままの 2 つの数を足すと待ち牌の合計数の数字の方が大きく，面子数を無視した評価となってしまうため，面子数と雀頭を合わせた数に 50 の重みをつけている．重み 50 は，ゲーム序盤の面子が少なく，多くの待ち牌で手が進む際に小さい重みであると面子と待ちで順序が逆転することから算出した．

4. 1. の仮に捨てる 1 枚を変更する．これを手牌 14 枚分繰り返し，最も評価点数が多い 1 枚を選択し捨てる．

これを繰り返しテンパイとなった際，即座にリーチを宣言する．これは，1.3 節で述べた通り，リーチにより振り込みを避けるため対戦相手が降りることが多くなり振り込む危険性が下がること，その結果自分があがることができやすくなること，あがれた場合に裏ドラにより運が良ければ点数を稼ぐことができることという利点があるためである．実際に，テンパイ時は即座にリーチを宣言した方が良いとされている [4]．

3.2.2 鳴きの戦略

鳴きを考慮することで，門前で手を進めるよりもさらに素早いあがりが可能となる．ただし，門前でテンパイした際のリーチができなくなってしまうため，別の役をつけないとあがることができない．そこで，門前で役牌を鳴くことができる場合のみ鳴くこととする．1 度役牌を鳴いた後は，有効牌を次々に鳴いていき，あがりを目指す．聴牌したときは，聴牌を維持し，かつ待ち牌が増える場合には手牌の入れ替えを行い，それ以外のときは自摸切りをする．

3.2.3 降りの戦略

2.1 節で述べた通り，麻雀では相手のあがり牌を捨てロンあがりされてしまうと，あがり点数を全て支払う必要があり，とても痛手となる．そこで，ロンあがりを防ぐため，以下のどちらかを満たした時点で降りを行う．

- 相手がリーチした場合
- 残り牌が 15 枚以下となった場合
  局終盤でテンパイできていない場合は，相手が鳴きもしくはリーチせずにテンパイしている可能性があるため降りる．

なお，鳴きによりリーチなしでテンパイしている場合，降りの戦略が優先される．降りを始めたがテンパイしリーチできる状況であれば，リーチが優先される．

また，以下の順で優先して捨てるべき牌を決定している．

1. 相手の捨て牌もしくは相手がリーチ後に他のプレイヤーが捨てた牌
2. 相手の捨て牌のスジである牌
3. 既に場に 1 枚以上見えている字牌

スジについて

テンパイの際，待ち牌が多い方がいいため，一般的に順子の両側を待つことが多い [3]．スジはこの考えを

用いた安全牌の読み方である．

図 1

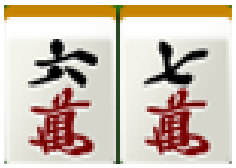

図 2

リーチしたプレイヤーが 5 を捨てているとする．すると，図 1 もしくは図 2 のような待ちである場合，あがり牌に 5 を含んでいるためフリテンとなる．つまり，5 を捨てていれば図 1 のもう 1 つの待ちである 2 と図 2 のもう 1 つの待ちである 8 はあたり牌でない可能性が高くなる．この 2 と 8 を「5 のスジ」という．なお，リーチしたプレイヤーが 2 を捨てている場合，図 1 の待ちはフリテンであるが，図 2 の待ちはフリテンでないため，2 のスジは 5 ではない．このスジの原理は 4・5・6 の牌を捨てている場合にその 3 つ隣の数字で発生する．

3.3 AI のプログラム

本研究で作成した AI のプログラムについて述べる．付録に作成した AI のプログラムのソースを示す．

3.4 プログラムの仕様

本 AI プログラムは [2] 内にある麻雀 AI インターフェイス「MJ_AI」クラスを継承して作られており，コンパイル後 jar ファイルへ変換し，[2] のディレクトリの AI フォルダに投下することでゲームプログラム実行の際に読み込まれ，機能する．図 3 にゲームプログラム内で AI プログラムが機能している様子を示す．

3.4.1 クラス MJAI_Test

クラス MJAI_Test は本研究で作成した AI プログラムの根幹部分である．以下にクラス MJAI_Test の主なメソッドについて述べる．図 4 にクラス MJAI_Test のクラス図を示す．

- int onSutehai(MJITehaiReader te, MJIHaiReader tsumohai)
  自分の手番の行動を選択する．どの牌を捨てるかを CalcSutehai で決め，リーチをするかを calcReachOrNot で判断する．すでにリーチしている場合はツモ切りのみをする．
- int calcSutehai(MJITehaiReader tehai, MJIHaiReader tsumohai)
  どの牌を捨てるかを決める．eval_tehai に 1 つの牌を捨てた時の手牌の評価を委譲する．また，降りの戦略の際は同時に eval_sutehai に 1 つの牌の危険度の評価を委譲する．2 つの評価を足し，これを手牌 14 枚分繰り返し，最も評価点の高い牌を選択し，返す．
- int eval_tehai(MJIHaiReader tehai_hai[], MJIHaiReader sutehai)
  手牌とひとつひとつの牌を評価する．手牌の評価は eval_tehai_connection に委譲し，ひとつひとつの牌の評価は eval_hai に委譲する．最終的に 2 つの評価点を足し合わせ，返す．
- int eval_hai(MJIHaiReader hai)

図 3　ゲームプログラム実行時の様子 (作成した AI はプレイヤー「Test」)

ひとつひとつの牌を評価する．面子を崩さずドラや赤ドラを組み込める場合は組み込むように評価点をプラスし，返す．

- void eval_tehai_connection(MJIHaiReader tehai_hai[], MJIHaiReader sutehai)
  手牌の評価を委譲する．手牌の配列を te_cnt 配列にコピーし，te_cnt 配列を仮の手牌の構築や入れ替えに使う．最後に eval_tehai_connection1 に評価を委譲する．
- int eval_tehai_connection1(int sutehai)
  手牌評価の根幹部分．はじめに残り牌を確認する．仮の次のツモ 34 種類を 1 種類ずつ面子数と雀頭，その時の待ちを eval_tehai_connectionA から参照する．最後に評価点を 3.2.1 節の評価関数で計算し，返す．
- void eval_tehai_connection1(boolean atama_flag, int mentsu_suu)
  面子数，雀頭，待ちを確認する．仮の手牌である te_cnt から面子と雀頭を牌が被らないように抽出する．抽出した数が最大であれば待ちを確認する．
- int eval_sutehai(MJIHaiReader hai)
  牌の危険度を 3.2.3 節の戦略をもとに計算する．降りの確認する順番ごとに重みをつけて，順番通り動作するようになっている．危険度を返す．降りを用いない戦略ではこのメソッドは使用しない．
- boolean calcReachOrNot(MJITehai te, MJIHaiReader tsumohai, int sutehai_x)
  リーチするかどうかを返す．はじめに面前であるか，テンパイであるか，フリテンでないかを確認し，満たしていればどんな時でもリーチを打つ．点数計算もこのメソッドで行っている．
- int onAction(int action, int player_no, int target_no, MJIHaiReader hai)
  他プレイヤーの手番での自分の行動を選択する．鳴きやロンを行うかの判断を CalcNaki に委譲し，返す．

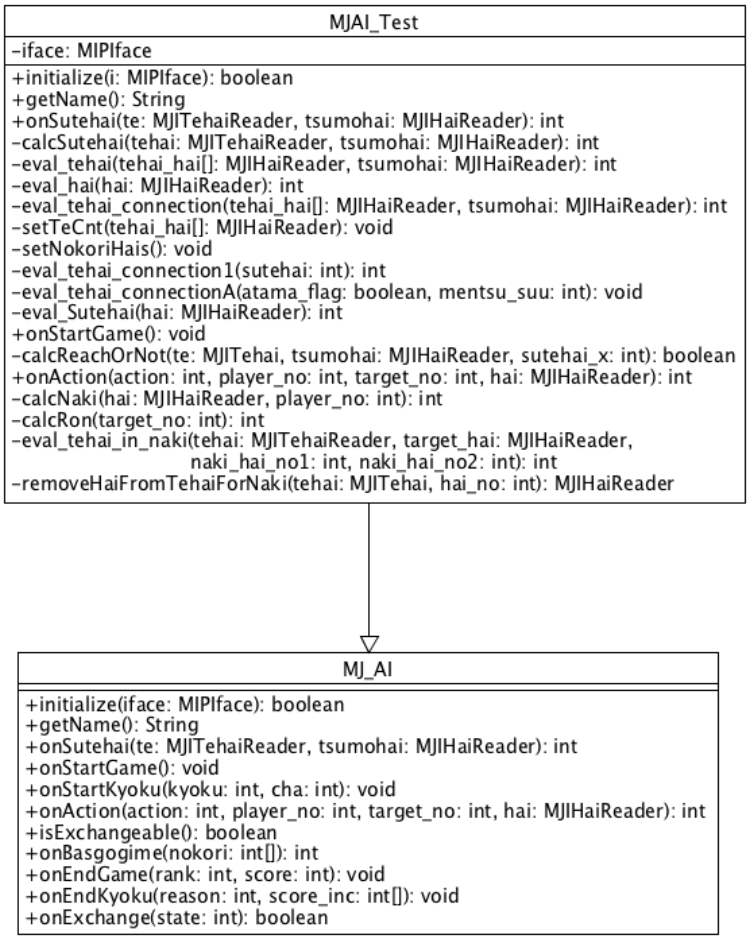

図 4　クラス MJAI_Test のクラス図

- int calcNaki(MJIHaiReader hai, int player_no)
  鳴くかどうかを決める．最初に，ロンするかどうかを calcRon で確認し，ロンでない場合は 3.2.2 節で述べた通り，自分が面前の場合は役牌を鳴ける場合のみ鳴き，その後は次々に鳴く．鳴く際，手牌の変化や評価は eval_tehai_in_naki に処理を委譲する．最後に鳴きの種類を返す．鳴きを用いない戦略ではロンするかどうかを委譲するのみとなる．
- int calcRon(int target_no)
  テンパイであるか，フリテンではないかを確認し，満たしていればロンをする．本研究ではテンパイの際，リーチしているか，役牌を鳴いているため，この 2 つの確認のみで良い．最後にロンするかどうかを返す．
- int eval_tehai_in_naki(MJITehaiReader tehai, MJIHaiReader target_hai, int naki_hai_no1, int naki_hai_no2)
  鳴きの際の副露の処理をする．また，手牌評価の際に副露した部分の評価点を加えるように操作し，評価点を返す．鳴きを用いない戦略ではこのメソッドは使用しない．

4　結果と考察

本研究では，ベースである 3.2.1 節の戦略 A，ベースに 3.2.2 節の鳴き戦略を加えた戦略 B，ベースに 3.2.3 節の降り戦略を加えた戦略 C，ベースに鳴き戦略と降り戦略の両方を加えた戦略 D を用意し，それぞれを [2] 上で付属の AI3 つと対戦させた．

4.1　結果

表 1 にそれぞれの戦略で東風戦を 300 回対戦させたデータを示す．

表 1　対戦結果

	戦略 A	戦略 B	戦略 C	戦略 D
局数	1690	1708	1686	1688
1 位率	30.7%	38.0%	30.0%	31.0%
4 位率	25.3%	17.3%	16.0%	16.7%
平均順位	2.36	2.18	2.31	2.29
和了率	22.4%	27.7%	19.8%	23.6%
放銃率	18.0%	17.0%	10.6%	10.8%

4.2　考察

各戦略の考察を以下に述べる．

- 戦略 A
  1 位率が高いが 4 位率も高く，素早くあがる戦略は鳴きや降りをしなければツモ運にある程度左右される一長一短の戦略であると示された．平均順位も他の戦略と比べると若干であるが劣っている．
- 戦略 B
  放銃率は戦略 A とあまり差がないが，和了率に大きく差が開き，鳴きによりさらに素早くあがれていることが示された．1 位率，平均順位が他の戦略と比べて抜けて高く，4 位率は降り戦略に迫っており，本研究で 1 番強い戦略となった．
- 戦略 C
  戦略 A よりも放銃率，4 位率が大きく下がっており，降りの戦略が動作していることが確認できる．ただし，戦略 A と比べて和了率が下がり，1 位率と平均順位はほとんど差が出なかった．
- 戦略 D
  4 位率，放銃率以外の全てのデータで戦略 B よりも大きく劣っており，4 位率に関しても戦略 B に降りを加えているにも関わらず同程度であることと，戦略 C と結果にほとんど違いが見られなかったことから，今回の降り戦略は鳴き戦略と相性がよくないと考えられる．

以下では，表 1 の結果について統計的に検証する．

AI 間に強弱の差が無ければ，1 位〜4 位になる確率はそれぞれ 25% である．そこで，1 位率および 4 位率を 25% と仮定したときに，統計上有意な差があるかを検証する．

勝率 p の勝負を N 回行った場合，標準偏差 s は以下の式で表される．

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

$p = 0.25$ と仮定すると，$N = 300$ ならば標準偏差は

$$\sqrt{300 * 0.25 * 0.75} = 7.50$$

となる．信頼区間 95% となるのは 1 位回数および 4 位回数の平均値からの差が $7.50 * 1.96 = 14.7$ となる区間である．したがって，1 位率および 4 位率が 25% ならば，300 試合すれば 95% の確率で 1 位回数および 4 位回数は 75 ± 14.7 ，確率として $25\% \pm 5\%$ に収まる．

つまり，今回の結果は全ての戦略で 1 位率が 3 割を超え，4 位率が戦略 A 以外で 2 割を切っていることから，戦略 B，戦略 C，戦略 D の強さは統計的に有意と言える．戦略 A も弱くはないが，4 位率の高さから強いとは言い切れない．

5　結論と今後の課題

本研究では素早くあがりを目指す戦略に加え，鳴きや降りの戦略を行う麻雀 AI の作成を行った．対戦結果より，ベースである素早くあがりを目指す戦略は強く，鳴きをすることでさらに強くなった．しかし，降り戦略は平均順位と勝率があまり良くなっておらず，まだまだ改良が必要である．

今後の課題として，前述の通り降り戦略の改良や鳴き戦略にタンヤオや三色同順などの比較的簡単な役を作ることができるような機能の追加が挙げられる．また，親ではあがり点が多いことと親の継続のため攻めるような戦略，子では無理をせず降りを重視する戦略をとるように，状況によって戦略をシフトできる AI を作成することが挙げられる．

謝辞

本研究を進めるにあたり，石水隆講師には計画の段階から実験内容，文献や書籍の提示など大変お世話になりました．1 年間ご指導いただきありがとうございました．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] 石畑恭平，まうじゃん的空間「まうじゃん for Java」，http://www.amy.hi-ho.ne.jp/ishihata/maujong/

[3] とつげき東北，おしえて！科学する麻雀，洋泉社，(2009).

[4] とつげき東北，科学する麻雀，講談社 (2004)

[5] 佐藤諒, 西村夏夫, 保木邦仁. 有効牌を数えて牌効率をあげる面前全ツッパ麻雀 AI の性能評価. 研究報告ゲーム情報学 (GI), 2014-GI-31, 11, pp. 1–6, (2014)
http://id.nii.ac.jp/1001/00099268/

[6] 麻雀 AI Microsoft Suphx が人間のトッププレイヤーに匹敵する成績を達成, Japan News Center, Mictosoft (2019/8/29)
https://news.microsoft.com/ja-jp/2019/08/29/190829-mahjong-ai-microsoft-suphx/

付録 A　本研究で作成した AI のソースコード

```
1  package jp.gr.java_conf.ishihata.maujong_ais;
2
3  import jp.gr.java_conf.ishihata.mj_ai.*;
4
5  public class MJAI_Test extends MJ_AI {
6      private MIPIface iface;
7
8      public boolean initialize(MIPIface i)
9      {
10         iface = i;
11         return true;
12     }
13
14     /**
15      * の名前AI
16      */
17     public String getName() {
18         return "Test";
19     }
20
21     /**
22      * 自分の番の行動
23      */
24     public int onSutehai(MJITehaiReader te, MJIHaiReader tsumohai)
25     {
26         if (iface.getAgariScore() > 0) return MJPIR_TSUMO;
27         if (iface.isKKHaiable()) return MJPIR_NAGASHI;
28
29         // リーチ中の場合ツモ切り
30         if (iface.isPlayerReached(0)) return MJPIR_SUTEHAI | 13;
31
32         MJITehai tehai = new MJITehai(te);
33
34         // 捨てる牌のインデックスを計算
35         int sutehai_index = calcSutehai(tehai, tsumohai);
36
37         // リーチするか判断
38         if (calcReachOrNot(tehai, tsumohai, sutehai_index))
39             return MJPIR_REACH | sutehai_index;
40         return MJPIR_SUTEHAI | sutehai_index;
41     }
42
```

```
43      /**
44       * 捨て牌を決める
45       */
46      private int calcSutehai(MJITehaiReader tehai, MJIHaiReader tsumohai)
47      {
48          int selected_hai = 0;    // 捨てる牌のインデックス
49          int score_max = -1;      // 最大評価値
50
51          // 手牌の配列を取得
52          MJIHaiReader tehai_hai[] = tehai.getTehai();
53
54          // ツモ牌を捨てた場合の評価
55          if (tsumohai != null){
56              // 手牌評価を計算
57              int tehai_score = eval_tehai(tehai_hai, tsumohai);降りるかどうか計算
58
59              //
60              int sutehai_score = eval_sutehai(tsumohai);
61
62              // 一度最大としておき、他の牌と比べる
63              score_max = tehai_score + sutehai_score;
64              selected_hai = 13;
65          }
66
67          // 持っている牌を一つずつ計算し、最大と比べる
68          for(int i = 0; i < tehai_hai.length; i ++){
69              // 捨てる牌
70              MJIHaiReader sutehai = tehai_hai[i];
71
72              // 一つ前の牌と同じ牌だったらスキップ
73              if (i > 0) if (sutehai.equals(tehai_hai[i - 1])) continue;
74
75              // この牌が捨てられない牌シャンテン数増加だったらスキップ()
76              if (! iface.isHaiThrowable(sutehai.getHaiNo())) continue;
77
78              // 捨てる牌をツモ牌と交換する
79              tehai_hai[i] = tsumohai;
80
81              // 手牌評価を計算
82              int tehai_score = eval_tehai(tehai_hai, sutehai);降りるかどうか計算
83
84              //
85              int sutehai_score = eval_sutehai(sutehai);
86
87              int score = tehai_score + sutehai_score;
```

```
88
89              // 比べる
90              if (score > score_max){
91                  selected_hai = i;
92                  score_max = score;
93              }
94
95              // 手牌情報を元の状態に戻す
96              tehai_hai[i] = sutehai;
97          }
98
99          return selected_hai;
100     }
101
102     /**
103      * 手牌、ひとつひとつの牌を評価
104      */
105     private int eval_tehai(MJIHaiReader tehai_hai[], MJIHaiReader sutehai)
106     {
107         // 牌の組み合わせに対する評価
108         int ret = eval_tehai_connection(tehai_hai, sutehai);
109
110         // ひとつひとつの牌に対する評価
111         for(int i = 0; i < tehai_hai.length; i ++){
112             ret += eval_hai(tehai_hai[i]);
113         }
114
115         return ret;
116     }
117
118     private final static int SCORE_KAZUHAI = 1; // 数牌に対する評価点
119     private final static int SCORE_DORA = 1;    // ドラに対する評価点
120     // 評価点に差をつけない
121
122     /**
123      * ドラと組み替えられる場合は替える
124      */
125     private int eval_hai(MJIHaiReader hai)
126     {
127         if (hai == null) return 0;
128
129         int ret = 0;
130
131         // ドラだったら評価点プラス
132         int[] doras = iface.getDora();
```

```
133             for(int i = 0; i < doras.length; i ++)
134                 if (hai.getHaiNo() == doras[i]) ret += SCORE_DORA;
135
136             // 赤ドラだったら評価点プラス
137             if (hai.hasAttribute(MJIHaiReader.ATTR_RED)) ret += SCORE_DORA;
138
139             return ret;
140         }
141
142         /**
143          * 評価点計算を委譲する
144          */
145         private int eval_tehai_connection(MJIHaiReader tehai_hai[], MJIHaiReader sutehai)
146         {
147             // 手牌配列から配列を構築te_cnt
148             setTeCnt(tehai_hai);
149
150             // 捨て牌しようとしている牌の牌番号
151             int sutehai_hai = -1;    // 何も捨てない場合は-1
152             if (sutehai != null) sutehai_hai = sutehai.getHaiNo();
153
154             return eval_tehai_connection1(sutehai_hai);
155         }
156
157         private int te_cnt[] = new int[34]; // 各牌の数
158
159         /**
160          * 手牌配列から配列を構築te_cnt
161          */
162         private void setTeCnt(MJIHaiReader tehai_hai[])
163         {
164             for(int i = 0; i < 34; i ++) te_cnt[i] = 0;
165
166             for(int i = 0; i < tehai_hai.length; i ++){
167                 if (tehai_hai[i] != null) te_cnt[tehai_hai[i].getHaiNo()] ++;
168             }
169         }
170
171         private final static int SCORE_TOITSU = 4;      // 対子の評価点
172         private final static int SCORE_KOTSU = 4;       // 刻子の評価点
173         private final static int SCORE_SHUNTSU_MACHI = 4;    // 順子待ちの評価点
174         private final static int SCORE_MENTSU = 50; // 面子の評価点
175         private final static int SCORE_ATAMA = 45;  // 雀頭の評価点
176         private final static int SCORE_MACHI = 1;    // 待ち牌の残り数に対する評価点対子、刻子、順
    子に差をつけない
```

```
177     //
178
179     private boolean my_anpai[] = new boolean[34];        // 自分の安全牌
180     private int nokori_hais[] = new int[34];             // 各牌の残り数
181     private int max_mentsu_suu;                          // 刻子、順子、雀頭の最大数
182     private boolean tehai_machi[] = new boolean[34];     // この手牌の待ち
183
184     // 各牌の残り数を数える処理
185     private void setNokoriHais()
186     {
187         for(int i = 0; i < 34; i ++){
188             nokori_hais[i] = 4 - iface.getVisibleHais(i) - te_cnt[i];
189         }
190     }
191
192     /**
193      * 現在のの状態を評価して評価点を返すte_cnt
194      */
195     private void eval_tehai_connectionA(boolean atama_flag, int mentsu_suu)
196     {
197         boolean nothing_flag = true;
198
199         // 面子と雀頭を取り出す
200         for(int p = 0; p < 34; p ++){
201             if (te_cnt[p] == 0) continue;
202             int c = te_cnt[p];
203
204             if (c >= 2 && !atama_flag) {
205                 te_cnt[p] -= 2;
206                 eval_tehai_connectionA(true, mentsu_suu + 1);
207                 nothing_flag = false;
208                 te_cnt[p] += 2;
209             }
210             if (c >= 3) {
211                 te_cnt[p] -= 3;
212                 eval_tehai_connectionA(atama_flag, mentsu_suu + 1);
213                 nothing_flag = false;
214                 te_cnt[p] += 3;
215             }
216
217             if (p < 27){
218                 int kazu = (p % 9) + 1;
219                 if (kazu < 8){
220                     if (te_cnt[p + 1] > 0 && te_cnt[p + 2] > 0){
221                         te_cnt[p] --; te_cnt[p + 1] --; te_cnt[p + 2] --;
```

```
222                     eval_tehai_connectionA(atama_flag, mentsu_suu + 1);
223                     nothing_flag = false;
224                     te_cnt[p] ++; te_cnt[p + 1] ++; te_cnt[p + 2] ++;
225                 }
226             }
227         }
228     }
229     if (! nothing_flag) return;
230
231     // 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
232     if (mentsu_suu < max_mentsu_suu) return;
233
234     // 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
235     // 最大面子数を更新して待ち牌情報をクリアする
236     if (mentsu_suu > max_mentsu_suu){
237         max_mentsu_suu = mentsu_suu;
238         for(int i = 0; i < 34; i ++) tehai_machi[i] = false;
239     }
240
241     // 残りの牌で待ちを確認する
242     for(int p = 0; p < 34; p ++){
243         if (te_cnt[p] == 0) continue;
244
245         // 対子
246         if (te_cnt[p] >= 2 && ! my_anpai[p]) tehai_machi[p] = true;
247
248         // まだ雀頭がなかったら、雀頭も待つ
249         else if (! atama_flag && ! my_anpai[p]) tehai_machi[p] = true;
250
251         // ペンチャン、カンチャン、両面待ち
252         if (p < 27){
253             int kazu = (p % 9) + 1;
254             // カンチャン
255             if (kazu < 8){
256                 if (te_cnt[p + 2] > 0){
257                     if (! my_anpai[p + 1]) {
258                         tehai_machi[p + 1] = true;
259                     }
260                 }
261             }
262             // ペンチャン、両面待ち
263             if (kazu < 9){
264                 if (te_cnt[p + 1] > 0){
265                     if (kazu > 1) {
266                         if (! my_anpai[p - 1]) {
```

```
267                                 tehai_machi[p - 1] = true;
268                             }
269                         }
270                         if (kazu < 8) {
271                             if (! my_anpai[p + 2]) {
272                                 tehai_machi[p + 2] = true;
273                             }
274                         }
275                     }
276                 }
277             }
278         }
279     }
280
281     /**
282      * 面子数と有効牌の残数を評価する
283      */
284     private int eval_tehai_connection1(int sutehai)
285     {
286         int ret = 0;     // 最終的な点数
287
288         // 次のツモまで先読みする
289         int all_nokori_hai = 0; // 全残り牌数
290         for(int i = 0; i < 34; i ++){
291             // の牌を増やすi
292             te_cnt[i] ++;
293
294             // 自分の安全牌をセットする
295             // また、この手牌の待ちを初期化する
296             for(int j = 0; j < 34; j ++){
297                 my_anpai[j] = iface.isHaiAnpai(0, j);
298                 tehai_machi[j] = false;
299             }
300             // 各牌の残りの数をセットする
301             setNokoriHais();
302             // 捨てようとしている牌も安全牌にして、その牌の残りの数も減らす
303             if (sutehai >= 0) {
304                 my_anpai[sutehai] = true;
305                 nokori_hais[sutehai] --;
306             }
307
308             // この牌の残りの数
309             int n = nokori_hais[i] + 1;
310             all_nokori_hai += n;
311
```

```
312                 if (n > 0) {
313                     // の牌が来たときの評価点を計算するi
314 
315                     // 刻子、順子、雀頭の最大数を初期化
316                     max_mentsu_suu = 0;
317 
318                     // 面子数、待ちを見つける
319                     eval_tehai_connectionA(false, 0);
320 
321                     // 刻子、順子、雀頭の評価点
322                     int score_mentsu = max_mentsu_suu * SCORE_MENTSU;
323 
324                     // 待ち牌に対する評価
325                     int score_machi = 0;
326                     for(int j = 0; j < 34; j ++){
327                         if (tehai_machi[j]) score_machi += nokori_hais[j] * SCORE_MACHI;
328                     }
329 
330                     // が来たときの手牌の評価点i
331                     int score = score_mentsu + score_machi;
332 
333                     // の牌の残りの数をかけて足していくi
334                     ret += n * score;
335                 }
336                 te_cnt[i] --;
337             }
338             return ret;
339         }
340 
341         private final static int SCORE_ANZEN = 1;              // 安全度に対する係数
342         private final static int SCORE_REACH_KIKEN = 1;        // リーチの危険度を表す係数
343 
344         /**
345          * 牌の危険度を評価する
346          */
347         private int eval_sutehai(MJIHaiReader hai)
348         {
349             int ds = 0;       // 危険度
350 
351             for(int i = 1; i < 4; i++){
352                 // アンパイなら危険度0
353                 if (iface.isHaiAnpai(i, hai.getHaiNo())) continue;
354 
355                 int plds = 0;         // このプレイヤーに対する危険度
356                 int hai_no = hai.getHaiNo();          // 捨て牌の牌番号
```

```
357
358             // アンパイでなければ、危険度アップ1
359             plds ++;
360
361             // スジのチェック
362             if (hai_no < 27){
363                 int kazu = (hai_no % 9) + 1;
364                 boolean fl = true;
365                 if (kazu > 3) if (!iface.isHaiAnpai(i, hai_no - 3)) fl = false;
366                 if (fl) if (kazu < 7) if (!iface.isHaiAnpai(i, hai_no + 3)) fl = false;
367                 // スジでなければ危険度アップ1
368                 if (!fl) plds ++;
369             } else {
370                 // 字牌の場合、初牌なら危険度アップ1
371                 if (iface.getVisibleHais(hai_no) == 0) plds ++;
372             }
373
374             // リーチだったら降りる
375             if (iface.isPlayerReached(i)) plds *= 500;
376
377             // リーチしてなくても残り牌以下なら降りる15
378             if (iface.getHaiRemain() < 16) plds *= 500;
379
380             ds += plds;
381         }
382
383         return (20 - ds) * SCORE_ANZEN;
384     }
385
386     private int i_oras_kyoku;    // オーラスの局
387
388     public void onStartGame()
389     {
390         // オーラスの局をセットしておく
391         if (iface.getRule(MIPIface.MJRL_NANNYU) == 0) i_oras_kyoku = 3;
392         else if (iface.getRule(MIPIface.MJRL_SHANYU) == 0) i_oras_kyoku = 7;
393         else i_oras_kyoku = 15;
394     }
395
396     /**
397      * リーチするかどうか
398      */
399     private boolean calcReachOrNot(MJITehai te, MJIHaiReader tsumohai, int sutehai_x)
400     {
401         int tsumohai_remain = iface.getHaiRemain();      // 残りのツモ牌の数
```

```
402
403             // 次のツモ牌がなければリーチできない
404             if (tsumohai_remain < 4) return false;
405
406             // 鳴いていないか？
407             if (te.getMinkans().length + te.getMinshuns().length + te.getMinkos().length > 0) retur
408
409             // 配列をセットしておくte_cnt
410             MJIHaiReader[] tehai_hai = te.getTehai();
411             setTeCnt(tehai_hai);
412
413             // 捨てようとしている牌の牌番号
414             int sutehai;
415
416             // 捨て牌したあとの手牌を用意する
417             if (sutehai_x < tehai_hai.length){
418                 sutehai = tehai_hai[sutehai_x].getHaiNo();
419                 te.removeHaiFromTehai(sutehai_x);
420                 te.addHaiToTehai(tsumohai);
421             } else {
422                 sutehai = tsumohai.getHaiNo();
423             }
424
425             // テンパイしているか確認
426             boolean machi[] = new boolean[34];
427             boolean b_tenpai = iface.getMachi(te, machi);
428
429             // テンパイでないならリーチしない
430             if (b_tenpai) return true;
431
432             // 待ち牌の残数を数え、あがった際の点数を計算
433             int machihai_remain = 0;
434             int agari_score = 0;
435             for(int i = 0; i < 34; i++){
436                 if (machi[i]) {
437                     // もしフリテンだったらリーチしない
438                     if (iface.isHaiAnpai(0, i) || i == sutehai) return false;
439
440                     // すでに場に出ている牌を数える
441                     int disp_num = iface.getVisibleHais(i) + te_cnt[i];
442                     if (tsumohai != null) if (tsumohai.getHaiNo() == i) disp_num ++;
443
444                     // もしこの牌がすべて出てしまっているなら次の牌へ
445                     if (disp_num >= 4) continue;
446
```

```
447                 machihai_remain += 4 - disp_num;
448
449                 // あがり点
450                 int sc = iface.getAgariScore(te, i);
451                 if (sc == 0 || sc < agari_score) agari_score = sc;
452             }
453         }
454
455         // 待ち牌が一つも残っていないならリーチしない
456         if (machihai_remain == 0) return false;
457
458         // リーチをかけた場合のあがり点
459         int reached_agari_score = agari_score * 2;
460
461         // もし役なしなら、ドラの数を数える
462         if (agari_score == 0){
463             int dora[] = iface.getDora();
464             int doras = 0;        // ドラの数
465             for(int i = 0; i < dora.length; i ++){
466                 doras += te_cnt[dora[i]];
467                 if (tsumohai != null) if (tsumohai.getHaiNo() == dora[i]) doras ++;
468                 if (sutehai == dora[i]) doras --;
469             }
470             // 赤ドラ
471             for(int i = 0; i < tehai_hai.length; i ++){
472                 if (tehai_hai[i].hasAttribute(MJIHaiReader.ATTR_RED)) doras ++;
473             }
474             // 暗槓のドラ
475             MJIHaiReader ankan_hai[][] = te.getAnkans();
476             for(int i = 0; i < ankan_hai.length; i ++){
477                 int hai = ankan_hai[i][0].getHaiNo();
478                 for(int j = 0; j < dora.length; j ++){
479                     if (hai == dora[j]) doras += 4;
480                 }
481                 // 赤ドラを探す
482                 for(int j = 0; j < 4; j ++){
483                     if (ankan_hai[i][j].hasAttribute(MJIHaiReader.ATTR_RED)) doras ++;
484                 }
485             }
486
487             // リーチした場合のあがり点を計算する
488             int han_suu = doras + 3;     // 飜数
489             reached_agari_score = 30 << han_suu;          // 符計算での基本点30
490             if (reached_agari_score >= 2000){
491                 // 満貫以上の計算
```

```
492                 if (han_suu < 8) reached_agari_score = 2000;            // 満貫
493                 else if (han_suu < 10) reached_agari_score = 3000;      // ハネ満
494                 else if (han_suu < 13) reached_agari_score = 4000;      // 倍満
495                 else if (han_suu < 15) reached_agari_score = 6000;      // 三倍満
496                 else reached_agari_score = 8000;                        // 数え役満
497             }
498             if (iface.getCha() == 0) reached_agari_score *= 6;
499             else reached_agari_score *= 4;
500             reached_agari_score += 90;
501         }
502         return true;
503     }
504 
505     /**
506      * 他家の番の自分の行動
507      */
508     public int onAction(int action, int player_no, int target_no, MJIHaiReader hai)
509     {
510         switch(action){
511         case MJPIR_REACH :     // リーチ
512         case MJPIR_SUTEHAI :  // 捨て牌
513             if (player_no != 0){
514                 // 自分以外の捨て牌だったら、鳴くか否かを判断
515                 return calcNaki(hai, player_no);
516             }
517             return 0;
518         case MJPIR_MINKAN :      // 大明槓加槓/
519             if (player_no != 0){     // 自分以外のカンだったら
520                 if (player_no == target_no){     // 加槓
521                     int sel_ron = calcRon(player_no);
522                     if (sel_ron == 2) return MJPIR_RON; // ロン
523                 }
524             }
525             return 0;
526         case MJPIR_ANKAN :       // 暗槓
527             return 0;
528         }
529         return 0;
530     }
531 
532     /**
533      * 鳴くかどうか決める
534      */
535     private int calcNaki(MJIHaiReader hai, int player_no)
536     {
```

```
537         // ロンするか？
538         int sel_ron = calcRon(player_no);
539         if (sel_ron == 2) return MJPIR_RON;      // ロン
540         if (sel_ron == 1) return 0;                          // テンパイしている場合は鳴かない
541
542         MJITehaiReader tehai = iface.getTehai();
543         MJIHaiReader tehai_hai[] = tehai.getTehai();
544         setTeCnt(tehai_hai);
545         int hai_no = hai.getHaiNo();     // 場に出た牌の牌番号
546
547         // すでに鳴いているか？
548         boolean menzen = tehai.getMinkans().length + tehai.getMinkos().length + tehai.getMinshu
549
550         // まず門前の場合の処理
551         if (menzen){
552             // 手に同一牌が枚あって、かつ字牌であること2
553             if (hai_no >= 27 && te_cnt[hai_no] == 2){
554                 // 字牌であってもオタ風ではダメ
555                 if (hai_no >= 31 || hai_no - 27 == iface.getCha() || hai_no - 27 == iface.getKy
556                     return MJPIR_PON;
557                 }
558             }
559             return 0;
560         }
561
562         // 現在の手牌の評価点を計算する
563         int max_score = eval_tehai(tehai_hai, null);
564
565         int ret = 0;
566
567         // すでに副露している場合は、価値のある牌だけ鳴く
568         // ポンの判定
569         if (te_cnt[hai_no] >= 2){
570             // この評価点に、鳴いた部分の評価点を加える
571             int sc = eval_tehai_in_naki(tehai, hai, hai_no, hai_no);
572             if (sc > max_score) {
573                 max_score = sc;
574                 ret = MJPIR_PON;
575             }
576         }
577
578         // チーの判定
579         // 上家以外の人が捨てた牌はチーできない
580         // また字牌はチーできない
581         if (player_no == 3 && hai_no < 27) {
```

```
582             int kazu = (hai_no % 9) + 1;          // 数牌の数の値
583
584             // 左端をチーする場合
585             if (kazu < 8) if (te_cnt[hai_no + 1] > 0 && te_cnt[hai_no + 2] > 0){
586                     int sc = eval_tehai_in_naki(tehai, hai, hai_no + 1, hai_no + 2);
587                     if (sc > max_score) {
588                         max_score = sc;
589                         ret = MJPIR_CHII1;
590                     }
591                 }
592             // 右端をチーする場合
593             if (kazu > 2) if (te_cnt[hai_no - 1] > 0 && te_cnt[hai_no - 2] > 0){
594                     int sc = eval_tehai_in_naki(tehai, hai, hai_no - 1, hai_no - 2);
595                     if (sc > max_score) {
596                         max_score = sc;
597                         ret = MJPIR_CHII2;
598                     }
599                 }
600             // 真ん中をチーする場合
601             if (kazu > 1 && kazu < 9) if (te_cnt[hai_no - 1] > 0 && te_cnt[hai_no + 1] > 0){
602                     int sc = eval_tehai_in_naki(tehai, hai, hai_no - 1, hai_no + 1);
603                     if (sc > max_score) {
604                         max_score = sc;
605                         ret = MJPIR_CHII3;
606                     }
607                 }
608         }
609         return ret;
610     }
611
612     /**
613      * ロンするか決める
614      * 戻り値ロンしない 0: テンパイだがロンしないフリテンなど1:() ロン2:
615      */
616     private int calcRon(int target_no)
617     {
618         // テンパイかどうか
619         boolean machi[] = new boolean[34];
620         boolean b_tenpai = iface.getMachi(machi);
621
622         // テンパイの場合、あがれるかチェック
623         if (b_tenpai){
624             int agari_score = iface.getAgariScore();     // あがり点
625             if (agari_score > 0){
626                 // フリテンチェック
```

```
627                 for(int i = 0; i < 34; i++){
628                     if (machi[i]) if (iface.isHaiAnpai(0, i)) return 1;
629                 }
630                 return 2;
631             }
632             return 1;
633         }
634         return 0;
635     }
636 
637     /**
638      * 副露とその評価をする
639      */
640     private int eval_tehai_in_naki(MJITehaiReader tehai, MJIHaiReader target_hai, int naki_hai_
641     {
642         MJITehai temp_tehai = new MJITehai(tehai);
643 
644         // 鳴く対象の牌2枚を探して手牌オブジェクトから取り除く
645         MJIHaiReader removed_hai1 = removeHaiFromTehaiForNaki(temp_tehai, naki_hai_no1);
646         MJIHaiReader removed_hai2 = removeHaiFromTehaiForNaki(temp_tehai, naki_hai_no2);
647 
648         // 鳴いたあとの捨て牌を決める
649         int sutehai_index = calcSutehai(temp_tehai, null);
650         MJIHaiReader temp_tehai_hai[] = temp_tehai.getTehai();
651         MJIHaiReader sutehai = temp_tehai_hai[sutehai_index];    // 捨てる牌
652         // その捨て牌を捨てたあとの手牌オブジェクトを生成
653         temp_tehai.removeHaiFromTehai(sutehai_index);
654         temp_tehai_hai = temp_tehai.getTehai();
655         // 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
656         int visible_hais_add[] = new int[2];
657         visible_hais_add[0] = naki_hai_no1;
658         visible_hais_add[1] = naki_hai_no2;
659         // 手牌評価
660         int temp_tehai_score = eval_tehai(temp_tehai_hai, sutehai);
661         visible_hais_add = null;
662         // この評価点に、鳴いた部分の評価点を加える
663         int sc = temp_tehai_score +
664             eval_hai(removed_hai1) + eval_hai(removed_hai2) +
665             eval_hai(target_hai) + SCORE_MENTSU;
666 
667         return sc;
668     }
669 
670     /**
671      * 手牌オブジェクトから、牌番号の牌を取り除き、取り除いた牌オブジェクトを返すtehaihai_no
```

```
672         */
673         private MJIHaiReader removeHaiFromTehaiForNaki(MJITehai tehai, int hai_no)
674         {
675             MJIHaiReader tehai_hai[] = tehai.getTehai();
676 
677             int index = 0;
678             for(int i = 0; i < tehai_hai.length; i ++){
679                 if (tehai_hai[i].getHaiNo() == hai_no){
680                     index = i;
681                     if (tehai_hai[i].hasAttribute(MJIHaiReader.ATTR_RED)) break;
682                 }
683             }
684 
685             MJIHaiReader hai = tehai_hai[index];
686             tehai.removeHaiFromTehai(index);
687             return hai;
688         }
689     }
```