卒業研究報告書

題目

麻雀ゲームにおけるAIの開発

指導教員

石水　隆　講師

報告者

08-1-037-0153

日高大地

近畿大学理工学部情報学科

平成25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。零和有限確定完全情報ゲームでは既知の結果より完全に解析を行い計算すれば必ず引き分け以上の結果が出ることが分かっている。しかし零和有限不確定非完全情報ゲームである麻雀は完全な解析が出来ず必勝法は存在しない。それを踏まえた上で、勝率を上げるために確率や統計を用いたアルゴリズムを開発研究している。実際にはアルゴリズムを作成しただけではなく強いアルゴリズムとして機能するように考える。最後に実戦を交えて開発された AI が本当に強いかを実証し検討する。

目次

1. 序論

1.1. 本研究の背景

近年,チェスやオセロ将棋囲碁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（以下,AI）の進歩が目まぐるしい。将棋やチェス等に代表されるボードゲームは、二人零和有限確定完全情報ゲームに分類される。零和とは、ゲーム終了時の全プレイヤーの点数合計が常に0となる。すなわち、相手の損がそのまま自分が利得となるゲームであり一方が勝てばもう片方は必ず負けとなる。有限とは、双方のプレイヤーが可能な手の組み合わせの総和が有限であるゲームである。確定とは、ゲームにおいて偶然の要素が入り込む余地がないゲームである。完全情報ゲームとは、ゲーム開始時から現在局面になるまでの全ての情報・選択を各プレイヤーが知ることができるゲームである。二人零和有限完全情報ゲームは、その性質上解析を行い易いため、ゲーム理論において様々な研究が今日までなされてきた。また、人工知能においても広く研究がなされている。

一方麻雀やトランプ、花札等に代表されるゲームは零和有限不確定非完全情報ゲームに分類される。不確定とは各プレイヤーの行動・着手だけでは次の局面が決まらないランダム要素が介入したゲームの事であり、多くはサイコロを用いたゲームに散見される。また非完全情報ゲームとは、全ての情報が与えられておらず一部の情報のみが各プレイヤーに与えられているゲームである。不確定非完全情報ゲームとは、その性質上ランダム要素を含むため先読みを行うことが難しく、これらの解析には困難を極める。

1.2. 零和有限不確定非完全情報ゲームに関する既知の結果

二人零和有限確定完全情報ゲームでは与えられた情報やこれまで培われてきた最善と考えられている一連の行動・着手を取る。それらはゲームにより定石・定跡・セオリーと呼ばれる。定石を基に完全に解析を行い計算をすれば必ず引き分け以上の必勝法が見つかることが分かっている。それはルール上、考えうる最適解に沿って常に行動できるからである。しかし零和有限不確定非完全情報ゲームではランダム要素が絡み、最適解に沿って行動しても必ずしも勝てるとは限らない。与えられた条件から確率や統計を駆使し少しでも勝率を上げるしかない。

1.3. 零和有限不確定非完全情報ゲームの既知のプログラム

不確定非完全情報ゲームの AI には主に大富豪やバックギャモンが挙げられる。1.2.節に上げたように完全には解析が出来ないがある程度の定石は存在することがある。しかし、不確定非完全情報ゲームにおいて単純な定石だけでは対戦者が AI の定石を見越した上で対策を取った場合それに対処できないことがある。ゲーム外においての読み合いとなりメタゲームとなりうる。TD ギャモン[5]ではそれらを多くの場数を踏み、自己強化することで克服した。TD ギャモンでは、試合中すぐには分からなかった行動の良否を、結果から学習を繰り返し次の試合以降は勝率の高い行動を取るようにし、人間のトッププレイヤーと互角の勝負が出来るようになった。麻雀の場合、バックギャモンとは違い4人で行う遊戯であり、また先読みを行なえば良い結果が生まれる訳ではない事から学習が難しい。麻雀 AI の作成は難しく、またルールの多様化や長期的な試合をしなければ実力が計りづらいことから対人戦およびプログラム同士の対戦はあまり進んでいない。

1.4. 本研究の目的

前節で述べた通り、不確定非完全情報ゲームは解析を行うことが難しく、また強いプログラムを作るのも難しい。そこで本研究ではランダム要素を加味した上で麻雀ゲームにおける思考を AI 化し、出来るだけ強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 インタフェイスを駆使して出来るだけ強い AI の開発を図る。また開発した AI が実際に強いのかどうか検証する為に、比較用に3つの AI とを対戦させ、その対戦成績を数値化し客観的に強い AI であるかどうかを検討する。

1.5. 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、本研究で作成する AI について戦略的に動作するクラスの概要を自分の手番、自分以外の手番に分けて述べる。4章では作成した AI と3つの AI との準備と対戦結果を考察する。5章ではこれまで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

2.1. 麻雀ゲームの概要

麻雀は主に四人で行うゲームである、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,以後時計回りにランダムに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,各プレイヤーは順番ぶ山から1枚の牌を引き(以下、ツモとする)、その中から1枚を捨てる行為を順番に繰り返し、四人の中で最も早く計14枚を予め決められた役に揃えることを目指す。

自分の手番中に,山から引いた牌を含む14枚の牌で定められた和了形が出来たらツモを宣言をする。または、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と自分の13枚の手牌を合わせると定められた和了形が出来ればロン(相手の捨て牌で和了)を宣言する。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とりをする。ツモで和了した場合は,和了したプレイヤーに対して他の3人が分担して得点を支払う。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牌を捨てた手番プレイヤーが1人で得点を支払う。得点は基本的に親は1.5倍多く得られるが、代わりに支払うべき得点も1.5倍となる。他のプレイヤーがロンで和了する手牌を捨ててしまうことを放銃と言う。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、基本的に親以外が和了すれば親を移動し次の局を始める。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。

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. 麻雀で使用する牌と基本的な和了形

麻雀で使用する牌は三種類(筒子・索子・萬子)の絵柄で1から9までの数字が書かれた27種類の数牌と東南西北・白發中と書かれた7種類の字牌、計34種類を使用する。各牌はそれぞれ4枚づつ存在する。図1に主な数牌と字牌を示す。数牌のうち1と9を老頭牌、2～8を中張牌と言う。また、字牌と老頭牌を合わせて么九牌(ヤオチューハイ)と言う。(以後、麻雀牌は便宜的に筒子は①②③④⑤⑥⑦⑧⑨、索子はアラビア数字で123456789、萬子は漢数字で一二三四五六七八九で表す)

図1:牌の種類　それぞれの数牌の老頭牌及び字牌

基本的な和了形には4つの「面子」と1つの「雀頭」を作らなければいけない(そうでない特殊な形がいくつかあるがここでは後述する理由により考えない)面子は数字が三枚順番で構成されている「順子」、同じ牌三枚で構成されている「刻子」、同じ牌四枚で構成されている「槓子」の三種類がある。雀頭は同じ牌二枚で構成されている「対子」がある。図2にそれぞれの形の一例を示す。

図2:左から順子・刻子・槓子・対子の一例

ただし、図3のように9から1にまたぐような順番は順子として認められない。このため老頭牌は中張牌よりも順子を構成しにくい。また字牌には数牌のような順番が存在しない。すなわち字牌の順子は存在しない。

図3:面子にならない一例

基本的な和了形以外の役に各数牌の老頭牌6種類と字牌7種類にその内どれかの1枚を加えた14枚で構成される「国士無双」、7つの対子で構成される「七対子」がある。国士無双は和了までの確率が低い上に、それによって得られる点数の期待値を考慮しても割に合わない。また七対子は役の性質上、鳴くことが出来ず必ず単騎待ち(1種類の牌のみの待ち)となる。また、他の役と複合しづらく点数においては七対子のみ別の計算式が採用され割に合わない。AI において考えても国士無双・七対子を目指すとなると途中からの戦略変更を採りづらくなるために本 AI では考えないことにする。

2.3. 麻雀の定石

麻雀の定石としては次のようなものがある。順子を構成しにくいことから、配牌からは字牌老頭牌を優先的に捨てた方がよい。また、和了できる牌が2種類になる両面待ちの形にすることが有効である。両面待ちとは、手牌中に完成面子3つと雀頭があり、残りの未完成面子が③④のように連続する2枚の中張牌である場合である。このとき両端の2種類(この場合②か⑤)のどちらかで順子が完成する。順子は刻子よりも完成し易いため、また多くの役とも複合し易いため、全ての面子を順子で構成する平和という役を狙うのが確率的にも点数的にも有効である。また先に他プレイヤーからリーチをされたら失点を防ぎたいため、放銃を避けた戦略を採った方がよい。

2.4. 麻雀プログラムで用いられる手法

麻雀ゲーム黎明期にはゲームの複雑さから配牌時点で和了形に近い状態にある、または手牌に近しい牌のみを優先的にツモるような試合自体を簡略化いわゆるイカサマを前提としたプログラミングが多く見受けられる。また CPU は捨て牌を選ぶ際に単純に役や持ち点を考慮せずに牌の組み合わせのみで行動することが多く、更に多面的に見た牌の処理が出来ない(例:②③③④④⑤と②②③③④④とでは後者の方では一盃口という役がつくが初期の AI では同じ2面子という扱いをする)。牌の組み合わせによる評価が出来るようになる AI が登場してきたが次に先を見据えた個々の牌の価値評価ができない問題に当たる(例:②④⑤⑥⑧の場合②と⑧を同じ評価とみなす。しかし実際は他の手牌の繋がりや残りの牌の数、ドラ・役を考えるため必ずしも同じ評価とは限らない)。その後、それらを克服した強い麻雀 AI が台頭し、麻雀におけるプレイスタイルの多様さと同じく麻雀 AI にも多くの打ち方が生まれた。

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」[2]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。ベースプログラムは。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)といった非合理的な要素を徹底的に排除し、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・流局をするか・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)である。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、点数はあまり考慮しない)を重視している。点数を考慮しない理由は他プレイヤーより先駆けて一番早くリーチをする事が長期的に見て和了率及び点数が高く、放銃率を下げる理由になるからである。

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。

- 自分が狙っている役を揃える

これは現在の手牌から、狙っている役を成立させるために必要な牌以外から捨て牌を選ぶ戦略である。この戦略を取ると狙った役が成立する確率が上がる。

- 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。

- ドラは持っておく

これは手牌中で、ドラに指定された牌がある場合、その牌以外から捨て牌を選ぶ戦略である。ドラとは、各局で1種類の牌がランダムに指定され、和了形にその牌が含んでいると点数が上がるが、ドラ単体では役にはならないボーナス牌である。ドラを持っておくことにより、和了による点数が上がる可能性がある。また同時に、他プレイヤーにドラを渡さないことにより、他プレイヤーの和了による点数を下げる可能性がある。

- 相手の危険牌を捨てない

危険牌とは、その牌を捨てると他プレイヤーがロンで和了する可能性が高い牌である。麻雀は1.1.節のように零和ゲームであり他プレイヤーの和了は出来るだけ避けなければならない。また2.1.節で述べたようにロンで和了した場合は、放銃したプレイヤーが全ての得点を支払わなければならないので大量失点の可能性が高い。危険牌は、プレイヤーの捨て牌からある程度の推察により絞り込むことができる。

麻雀のルールでは、自分が捨てた牌と同じ種類の牌に対してロンをすることはできない。従って相手の捨て牌は現物と呼ばれ、絶対にロンをされる危険がない安全牌なる。待ちについては2.3.節で述べたように両面待ちにすることが多いが、待ち牌となる可能性のある種類の内どれか一種類を捨てていると、その捨て牌を含む待ちではロンをすることができなくなることをフリテンと言う。このため、数牌の捨て牌の上下2つ飛ばした牌はスジと呼ばれ危険度が低い牌となる。字牌、老頭牌は性質上手牌から浮いてしまう事が多く危険度が低く、逆に中張牌

は和了形を多く作り易く、手牌に組み込まれる事が多い為に危険度を高くする。ドラは和了した場合点数が上がるため、ドラを捨てるのはロンや鳴かれることにより他のプレイヤーの点数を高くしてしまう可能性があるので、ドラも危険牌となりうる。

- 自分の狙っている役をバレないようにする

前述のように、各プレイヤーは危険牌を捨てる行為を避けようとする。あるプレイヤーが狙っている役がバレると、その危険牌も絞り込み易くなる。そこで、狙っている役がバレないように捨てることで他プレイヤーの放銃を誘い,危険牌を捨てさせる確率が上がる。

捨て牌の評価

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。

本研究で用いた手牌の評価値 E_{just} は以下の式により導き出される.

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

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

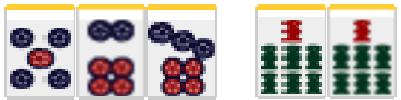
手牌の評価値は、手牌中の完成面子数と未完成面子数及び未完成面子に対する待ち牌が来る確率を求める。次にツモる可能性のある牌の種類およびその確率を、手牌およびプレイヤーが目視出来る場の情報(捨て牌・ポン・チー・カンされた牌およびそれに付随している牌・ドラ表示牌)から求める。次に仮に手牌からある牌を一つ捨てた場合、次にツモる可能性のある牌の種類とその確率から、各牌をツモった場合の手牌の評価値の期待値をそれぞれの牌を捨てた場合の評価値から計算し、最も評価値を高くする牌を捨てる。

以下図4に示す手牌例を参考に手牌の評価アルゴリズムが実際にどう機能しているかを述べる。図4の例では⑤⑥⑦で順子、東と中で刻子がそれぞれ完成している。東と中はそれぞれ独立した刻子であり、他のどの牌が来ても繋がりを持たないことからここでは除外して残りの7牌で考える。まず各牌について、手牌および場の情報からその牌が残り何枚あるかを求めたときに、残った手牌中の完成面子と雀頭を除く牌を用いて完成面子および雀頭を作るのに必要な待ち牌及び待ち牌の残り数を求める。実際のプログラムでは場に見えている牌の数は減らして計算しているが以下では全ての牌がまだ出ていないものとする。手牌例からシャンテン数(和了まで有効牌があと何枚必要か)が減る牌を列挙する。

図4: 手牌の一例

- 仮にツモる牌 No.1

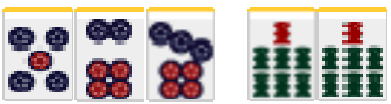
完成面子と雀頭



未完成面子

待ち牌　残り1種4枚

- 仮にツモる牌 No.2



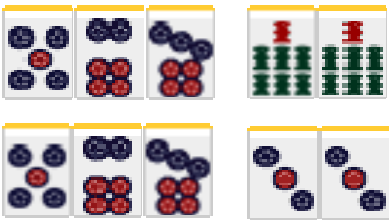
完成面子と雀頭

未完成面子

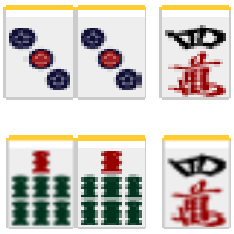
待ち牌 残り2種8枚

- 仮にツモる牌 No.3

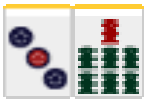
完成面子と雀頭



未完成面子

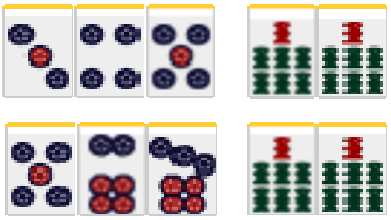


待ち牌　残り2種4枚

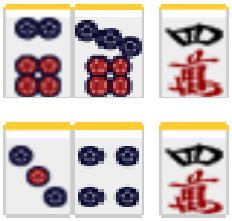


- 仮にツモる牌 No.4

完成面子と雀頭



未完成面子



待ち牌　残り3種11枚

- 仮にツモる牌 No.5

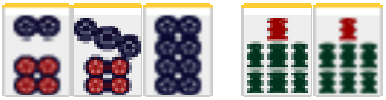
完成面子と雀頭

未完成面子

待ち牌　残り1種4枚

- 仮にツモる牌 No.6

完成面子と雀頭

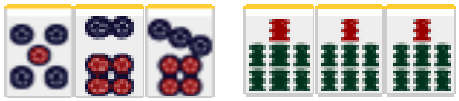


未完成面子

待ち牌　残り1種4枚

- 仮にツモる牌 No.7

完成面子　　未完成雀頭　　待ち牌　残り2種6枚



- 仮にツモる牌 No.8

完成面子と雀頭　　未完成面子　　待ち牌　残り1種4枚



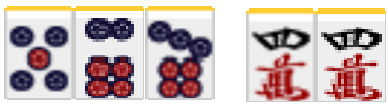
- 仮にツモる牌 No.9

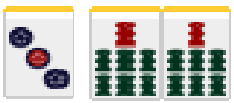
完成面子と雀頭　　未完成面子　　待ち牌　残り2種8枚



- 仮にツモる牌 No.10

完成面子と雀頭　　未完成面子　　待ち牌　残り2種4枚




- 仮にツモる牌 No.11

完成面子と雀頭　　未完成面子　　待ち牌　残り2種8枚



- 仮にツモる牌 No.12

完成面子と雀頭

未完成面子

待ち牌　残り1種4枚

以上の列挙から未完成面子に分類される単独牌は「四」(No.1～7)と「③」(No.7～12)のみである。No.7に関してはどちらも同じ確率で相手が雀頭になる可能性があるので待ち牌は3枚づつで考える。全ての待ち牌を合計すると「四」は38枚、「③」は31枚であり、両者を比べると「③」の方が面子に組み込まれる確率が高く「四」を捨てた方が損失が小さいことが分かる。よってこの手牌の場合は「四」を捨てるという結論に至る。

3.2.2. 戦略の選択

3.2.1.節で挙げた戦略のうち、どの戦略を重視するべきかは状況により異なる。麻雀の戦略は、大きく分けて勝ちを狙う戦略と負けを避ける戦略に分けられる。勝ちを狙う戦略は、自分が和了することを目指すものである。この戦略では、自分が狙う役を揃える、面子や対子が出来ている牌を残す、といった戦略を用いて、自分が和了する確率を上げ和了した場合に得られる点数を高くすることを目指す。一方、負けを避ける戦略では、他のプレイヤーの危険牌を捨てない戦略を採り、できるだけ放銃による失点をしないようにする。どちらの戦略を採るかは自分が今狙っている手役の高さ、現在の順目等から決定する。

3.2.3. 自分以外の手番での戦略

自分以外の手番において思考を要する行為は「手番プレイヤーの捨て牌に対してポン・チー・カン（以下、鳴く）をするか」である。それらを判断するアルゴリズムとして現在の状況（点差や狙っている手、形式テンパイのみを狙う）を考え、評価関数を満たしていれば鳴くようにする。

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。

- void set_te_cnt()
  set_te_cnt() は手牌を読み取る準備を行うメソッドである。

- int onSutehai()
  onSutehai() は捨て牌を捨てる前に判断をするメソッドである。リーチしているなら自動的にツモ牌を捨てる。一度も鳴いていないなら後述する CalcReachOrNot()メソッドを参照し、点数や現在順目を考えてリーチするかを判断する。

- int calc_sutehai」()
  calc_sutehai() は捨て牌を決めるメソッドである。捨て牌や手牌の評価関数を読み取り捨て牌を判断する。牌の評価関数は後述の eval_hai eval_tehai() メソッドで参照する。

- int eval_tehai()
  eval_tehai() は手牌の評価値を計算するメソッドである。後述する eval_hai() メソッドと eval_tehai_connection() メソッドで得られた結果を合算し評価する。

- int eval_hai()
  eval_hai() は個々の牌をそれぞれ評価するメソッドである。老頭牌や字牌は低く、中張牌・ドラは高い評価を

する。

- int eval_tehai_connection(), int eval_tehai_connection_1_sub(), int eval_tehai_connection_2()
  これらはそれぞれ3.2.1.節に述べた通りの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()
  evak_stehai() は相手の捨て牌を見るメソッドである。相手の捨て牌の情報から危険牌を捨てないようにする。リーチした相手には危険度を増す。スジや現物かどうかをチェックする。

- boolean CalcReachOrNot()
  CalcReachOrNot() はリーチをするか判断するメソッドである。山の残り数や鳴いていないか、テンパイしているかフリテンしているか待ち牌が残っているかをチェックし、ある程度の点差がついていればリーチしない。またオーラスでは和了した時の点数を計算し、リーチの必要性を考える。リーチの細かな条件は[3]に基いている。

- int calcNaki()
  calcNaki() は鳴き関するメソッドである。字牌であってオタ風であれば鳴かない。ドラは極力鳴く、残り山牌が少なければ形式テンパイを狙うために鳴く。特にオーラスでは和了した時の点数を計算しロンをするか考える。

4. 結果・考察

本研究では、前章で述べた AI および比較用に3つの AI(AI$_1$:鳴かずに手作りする / AI$_2$:平和役を最速で目指す / AI$_3$:危険を察知したら振り込まないようにベタオリする)を用意し、各 AI と対戦を行なった。

4.1. 対戦結果

表1に300局を対局したデータを示す。表1中の和了率、放銃率は、対戦中各 AI がそれぞれ和了、放銃した確率であり、1位率、4位率は、半荘を一試合としたときにそれぞれ1位、4位になった確率である。また表1から予想される長期的な平均順位を表2に示す。表2の各項目は予想順位の平均(±標準偏差)を示す。予想平均順位は以下の2つの指標により導かれる。

指標1 : 2.63 − 2.40*和了率 + 2.69*放銃率
指標2 : 2.52 − 1.62*1位率 + 1.52*4位率

表1　対戦成績(対戦局数300)

	AI	AI$_1$	AI$_2$	AI$_3$
和了率	26.4%	17.9%	20.4%	17.9%
放銃率	13.7%	17.8%	16.8%	10.8%
1位率	38.2%	19.6%	20.9%	21.3%
4位率	15.6%	33.2%	28.6%	22.3%

表2　予想平均順位

	AI	AI$_1$	AI$_2$	AI$_3$
指標1	2.36(±0.12)	2.68(±0.12)	2.59(±0.12)	2.49(±0.12)
指標2	2.14(±0.22)	2.71(±0.22)	2.61(±0.22)	2.51(±0.22)

4.2. 考察

標準的な成績であれば平均順位は2.5位に収束する事を加味すると本 AI が表2より指標1において2.36位,指標2においては2.14位という成績をみると圧倒的に強い事が示された.放銃率ではベタオリを重視する AI$_3$に劣っているが和了率で優っているので大局的に見れば本 AI の方が勝つことが多い.それぞれ3つの AI は決められた命令には従うが複合的にバランス良く考えられた本 AI の強さが目立った.

5. 結論と今後の課題

本研究では、麻雀 AI の開発を行った。AI 同士の対戦結果より、非常に強い AI の開発が出来たことが示される。しかし麻雀は不確定非完全情報ゲームであり、1.2.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。

今後の課題として麻雀戦略における膨大な癖やタイプをデータとして事前に読み込ませ、対戦中に相手のパターンを察知し先を読み、その上で相手に合わせた戦略にシフトできる、より強い AI を開発する事が挙げられる。また数試合程度では強さは測れず数百、数千試合をこなしてある程度の強さが測れるのでその簡略化も同時に求められる。本 AI では予め提供されているプログラム上でしか起動しなかったが全ての戦況(点数・場・手牌等)を入力すれば評価の高い判断が出来るような汎用性の高いプログラムが望まれる。

参考文献

[1] 石畑恭平: コンピュータ麻雀のアルゴリズム,IO BOOKS,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: 科学する麻雀,講談社現代新書(2004)
[4] 人工知能学会: 人工知能の話題 TDギャモン (TD-Gammon),
http://www.ai-gakkai.or.jp/whatsai/AItopics4.html
[5] Gerald Tesauro: TD-Gammon,Communications of the ACM(1995),
http://www.research.ibm.com/massive/tdl.html
[6] 松原仁: ゲーム情報学の中の不完全情報ゲームの位置づけと現状, パネル討論「不完全情報ゲームにおける競技性について」, 第6 回エンターテイメントと認知科学シンポジウム, (2012),
http://entcog.c.ooco.jp/entcog/event/20120318/matsubara.pdf
[7] 東育生, 橋本剛, 飯田弘之: 完全情報ゲームと不完全情報ゲームの戦略的架け橋 麻雀を題材として, 情報処理学会研究報告ゲーム情報学(GI), Vol2000-GI-003, pp.65-70 (2000),
http://id.nii.ac.jp/1001/00058644/

付録 麻雀 AI のソースプログラム

以下に本研究で作成した麻雀 AI のソースプログラムを示す。

```
import jp.gr.java_conf.ishihata.mj_ai.*;

public class AIP extends jp.gr.java_conf.ishihata.mj_ai.MJ_AI {
	private int[] te_cnt = new int[34];
	private boolean[] machi = new boolean[34];
	private boolean menzen = true;
	private boolean nakiok_flag = false;
	private boolean tenpai_flag = false;
	private int tehai_score = 0;
	private int sthai = -1;

	//インスタンスの呼び出し
	private MIPIface iface;
	public boolean initialize(MIPIface i){
		iface = i;
		return true;
	}

	//AIの名前
	public String getName(){
		return "メインAI";
	}

	//評価関数
	private final static int SCORE_MENTSU = 50;
	private final static int SCORE_MACHI = 1;

	//手牌を読み取る準備クラス
	private void set_te_cnt(MJITehaiReader tehai){
		int i;
		for(i=0;i<34;i++) te_cnt[i] = 0;
		MJIHaiReader[] hais = tehai.getTehai();
		for(i=0;i<hais.length;i++) te_cnt[hais[i].getHaiNo()]++;
	}

	//捨て牌を決める前の判断をする
	public int onSutehai(MJITehaiReader te, MJIHaiReader tsumohai){
		if (tsumohai != null) if (machi[tsumohai.getHaiNo()]) {
			return MJPIR_TSUMO;
		}
		//自分がリーチしているならツモ牌を自動で捨てる
		if (iface.isPlayerReached(0)) return MJPIR_SUTEHAI | 13;
		if (iface.isKKHaiable()) return MJPIR_NAGASHI; //9種9牌なら流す

		//手牌を読み取る
		MJITehai tehai = new MJITehai(te);
		set_te_cnt(tehai);

		if (tsumohai != null) te_cnt[tsumohai.getHaiNo()]++;
		boolean[] mc = new boolean[34];
		int rchk = MJPIR_SUTEHAI, i;

		int hai = calc_sutehai(tehai, tsumohai);
		MJIHaiReader[] tehai_hais = tehai.getTehai();

		//面前ならリーチするか考える
		if (menzen){
```

```
                    if (hai<tehai_hais.length){
                        tehai.removeHaiFromTehai(hai);
                        tehai.addHaiToTehai(new MJIHai(tsumohai));
                    }
                    iface.getMachi(tehai, mc);
                    for(i=0;i<34;i++){
                        if (mc[i]) {
                            if (te_cnt[i]+iface.getVisibleHais(i)<4){
                                if (iface.getHaiRemain()>60) {
                                    rchk = MJPIR_REACH; break;
                                }
                                if (iface.getAgariScore(tehai,i)<=0){
                                    rchk = MJPIR_REACH; break;
                                }
                            }
                        }
                    }
            }

            if (hai<tehai_hais.length){
                te_cnt[tehai_hais[hai].getHaiNo()]--;
            } else te_cnt[tsumohai.getHaiNo()]--;

            return hai|rchk;
    }

    //手配の評価を計算するメソッド
            private int eval_tehai(MJIHaiReader tehai_hai[],MJIHaiReader sutehai){
                //牌の組み合わせによる評価関数
                int ret = eval_tehai_connection(tehai_hai,sutehai);

                //一つ一つの牌に対する評価関数
                for(int i = 0; i < tehai_hai.length; i++){
                    //ret += eval_hai(tehai_hai[i]) //
                    ret += eval_hai(i) + te_cnt[i];
                }
                return ret;
            }

    //牌の単独評価をする
    private int eval_hai(int param){
            int i = 0;
            if (param < 27)
              i++;
            else if ((this.iface.getVisibleHais(param) < 2)
                && ((param > 30) || (param == this.iface.getCha() + 27)
                    || (param == this.iface.getKyoku() / 4 + 27)))
              i++;
            int[] arrayOfInt = this.iface.getDora();
            for (int j = 0; j < arrayOfInt.length; j++)
              if (param == arrayOfInt[j])
                i++;
            return i;
          }

    //牌の組み合わせによる評価関数
    int te_cnt[] = new int[34];//牌の種類が34種なので各牌の枚数
    private void setTeCnt(MJIHaiReader tehai_hai[]){
            for(int i = 0; i < 34; i++);
            for(int i = 0; i < tehai_hai.length; i ++){
                    te_cnt[tehai_hai[i].getHaiNo()] ++;
            }
```

```
}

//捨て牌選択の戦略を決める
private int eval_tehai_connection(MJIHaiReader tehai_hai[],MJIHaiReader sutehai){
		setTeCnt(tehai_hai);
		int sutehai_hai = -1;
		if(sutehai != null)sutehai_hai = sutehai.getHaiNo();

		return eval_tehai_connection_2(sutehai_hai);
}

//手牌を評価するメソッド
int score = 0;
private boolean my_anpai[] = new boolean[34];
private int max_mentsu_suu;
private boolean tehai_machi[] = new boolean[34];
private int nokori_hais[] = new int[34]; //各牌の残り枚数

//手牌の組み合わせを見て外れた牌から待ちを予想する
private void eval_tehai_connection_1_sub(boolean atama_flag,int mentsu_suu){
		boolean nothing_flag = true;
		for(int p = 0; p < 34; p++){
			if(te_cnt[p] == 0) continue;
			int c = te_cnt[p];

			if(c >= 2 && !atama_flag){
				te_cnt[p] -=2;
				eval_tehai_connection_1_sub(true,mentsu_suu + 1);
				nothing_flag = false;
				te_cnt[p] += 3;
			}
			if(p >= 3){
				te_cnt[p] -=3;
				eval_tehai_connection_1_sub(atama_flag,mentsu_suu + 1);
				nothing_flag = false;
				te_cnt[3] += 3;
			}

			if(p < 27){
				int suu = (p % 9) + 1;
				if(suu < 8){
					if(te_cnt[p + 1] > 0 && te_cnt[p + 2]> 0){
						te_cnt[p]--; te_cnt[p + 1]--; te_cnt[p + 2]--;
						eval_tehai_connection_1_sub(atama_flag,mentsu_suu + 1);
					}
				}
			}
		}
		if (! nothing_flag) return;

		//面子数が最大面子数よりも少ないならここで終わり
		if(mentsu_suu < max_mentsu_suu) return;
		//面子数が最大面子すうよりも大きかったら最大面子数を更新して
		//牌情報を更新してクリアする
		if(mentsu_suu > max_mentsu_suu){
			max_mentsu_suu = mentsu_suu;
			for(int i = 0;i < 34; i++)
				tehai_machi[i] = false;
		}

		//残り牌で待ちを確定させる
		for(int p = 0; p < 34; p++){
```

```
                        if(te_cnt[p] == 0) continue;

                        //対子
                        if(te_cnt[p] > 2 && ! my_anpai[p])
                                tehai_machi[p] = true;
                        //雀頭がなければ雀頭も待つ
                        else if(! atama_flag && ! my_anpai[p])
                                tehai_machi[p] = true;

                        //ペンチャン、カンチャン、二面待ちに対して
                        if(p > 27){
                                int suu = (p % 9) + 1;
                        //カンチャン
                                if (suu < 8){
                                        if (te_cnt[p+ 2] > 0){
                                                if(! my_anpai[p + 1]){
                                                        tehai_machi[p + 1] = true;
                                                }
                                        }
                                }
                                //ペンチャン、二面待ち
                                if(suu < 9){
                                        if (te_cnt[p + 1] > 0){
                                                if(suu > 1){
                                                        if(! my_anpai[p - 1]){
                                                                tehai_machi[p - 1] = true;
                                                        }
                                                }
                                                if(suu < 8){
                                                        if(! my_anpai[p + 2]){
                                                                tehai_machi[p + 2] = true;
                                                        }
                                                }
                                        }
                                }
                        }
                }
        }

        private int eval_tehai_connection_2(int sutehai){
                //最終的な評価関数
                int ret = 0;
                //次のツモを予測する
                int all_nokori_hai = 0;
                for(int i = 0; i < 34; i++){
                        te_cnt[i] ++;
                        for(int j = 0; j < 34; j++){
                                my_anpai[j] = iface.isHaiAnpai(0,j);
                                tehai_machi[j] = false;
                        }
                        setNokoriHais();
                        //捨てようとしている牌を安全牌にして残り数から1減らす
                        if(sutehai >= 0){
                                my_anpai[sutehai] = true;
                                nokori_hais[sutehai] --;
                        }
                        //この牌の残り数を出す
                        int n = nokori_hais[i] +1;
                        all_nokori_hai += n;

                        if(n > 0){
                                //この牌が来た時の評価関数を出す,組み合わせの最大数を初期化
```

```
                                max_mentsu_suu = 0;
                                eval_tehai_connection_1_sub(false,0);

                                int score_mentsu = max_mentsu_suu * SCORE_MENTSU;

                                int score_machi = 0;
                                for(int j = 0; j < 34;j++){
                                        if(tehai_machi[j])
                                                score_machi *= nokori_hais[j] * SCORE_MACHI;
                                }

                                int score = score_mentsu + score_machi;
                                ret += n * score;
                        }
                        te_cnt[i] --;
                }
                //残り牌の数で割り、評価関数を加重平均を出す
                ret /= all_nokori_hai;
                return ret;
        }

        //相手の捨て牌を見て考えるアルゴリズム
        private int eval_sutehai(MJIHaiReader hai){
                int dn = 0; //危険度danger number
                int playerdn = 0; //そのプレイヤーの危険度
                int hai_no = hai.getHaiNo(); //捨て牌の牌番号

                for(int i = 0; i < 4; i++){
                        if (iface.isHaiAnpai(i,hai_no)) continue;

                playerdn ++;

                if(hai_no > 27){
                        int kazu = (hai_no % 9) + 1;
                        boolean fl = true;
                //スジ牌のチェック
                        if(kazu > 3) if(!iface.isHaiAnpai(i,hai_no - 3))
                                fl = false;
                        if(kazu > 3) if(!iface.isHaiAnpai(i,hai_no + 3))
                                fl = false;
                //スジがないなら危険度を上げる
                        if(!fl) { playerdn ++;
                        } else {
                                if(iface.getVisibleHais(hai_no) == 0) playerdn ++;
                        }
                //リーチ者には危険度を極端に上げる
                        if (iface.isPlayerReached(i)) playerdn *= 100;
                        return dn += playerdn;
                }
                return 20 - dn;
        }

        public boolean calcReachOrNot(MJITehai te, MJIHaiReader tsumohai, int sutehai_x){
                nt tsumohai_remain = iface.getHaiRemain();
                int sutehai;
                int machihai_remain = 0;
                int agari_score= 0;

                if(tsumohai_remain < 4)
                        return false; //山が残り4牌を切るとリーチ出来ない
                if(te.getMinkans().length + te.getMinshuns().length + te.getMinkos().length > 0)
                        return false; //カン、チー、ポンの鳴きをしていればリーチ出来ない
```

```
//手牌をセットする
MJIHaiReader[] tehai_hai = te.getTehai();
setTeCnt(tehai_hai);

//捨て牌をした後の手牌をセットする
if(sutehai_x < tehai_hai.length){
		sutehai = tehai_hai[sutehai_x].getHaiNo();
		te.removeHaiFromTehai(sutehai_x);
		te.addHaiToTehai(tsumohai);
} else {
		sutehai = tsumohai.getHaiNo();
}

//テンパイをしているかチェック
boolean[] machi = new boolean[34];
boolean b_tenpai = iface.getMachi(te,machi);

//テンパイでないならリーチしない
if(!b_tenpai) return false;

//全ての対戦相手を調べて12000点以上離れていたらリーチしない
{
int my_score = iface.getScore(0);
int i = 3; //3は上家,2は対面,1は下家
for(; i > 0; i--){
		if (my_score < iface.getScore(i) + 12000) break;
}
if(i == 0) {
		return false;
}

//フリテンならリーチしない
for(int i = 0; i < 34; i++){
		if(machi[1]){
			if(iface.isHaiAnpai(0,1) || i == sutehai) return false;
//場に出ている数と手牌にある数から残り牌をカウントする
int left_num = iface.getVisibleHais(i) + te_cnt[i];
		if(tsumohai != null)
			if(tsumohai.getHaiNo() == i)
				left_num ++; //ツモッた牌は出てしまった牌として加算する
		if(left_num >= 4) continue; //4枚とも出てしまったら次の待ち牌候補へ移る
		machihai_remain += 4 - left_num; //残り牌
		//あがり点数を出す
		int sc = iface.getAgariScore(te,i);
		if (sc == 0 || sc < agari_score)
			agari_score = sc;
		}
}
//待ち牌が全て切れていたらリーチしない
if (machihai_remain == 0)
		return false;
//ダマテンで7700点以上ならリーチしない
if(agari_score > 7700)
		return false;
//オーラスでの判断
//トップと自分との点差を計算
if(iface.getKyoku() == i_oras_kyoku){
		int top_score = iface.getScore(1);
		for(int i = 2;i < 4; i++){
			int sc = iface.getScore(i);
			if(sc > top_score) top_score = sc;
```

```
}
int my_score = iface.getScore(0); //自分の持ち点
int gap = top_score - my_score; //点差
//自分がトップならリーチをかけない、リーチをしなくても逆転出来るならリーチしない
if(gap < agari_score)return false;
		return true;
}

//リーチをかけた場合のあがり点 概算
int reached_agari_score = agari_score * 2;
//点数が0ならドラを数える
if(agari_score == 0){
		int dora[] = iface.getDora();
		int doras = 0;
		for(int i = 0; i < dora.length; i++){
			doras += te_cnt[dora[i]];
		if(tsumohai != null) if(tsumohai.getHaiNo() == dora[i]) doras ++;
		if(sutehai == dora[i]) doras --;
}
		//赤ドラを探す
		for(int i = 0; i < tehai_hai.length; i++){
			if(tehai_hai[i].hasAttribute(MJIHaiReader.ATTR_RED)) doras ++;
			}
		//暗カン内のドラも探す
		MJIHaiReader ankan_hai[][] = te.getAnkans();
		for(int i = 0;i < ankan_hai.length;i++){
			int hai = ankan_hai[i][0].getHaiNo();
			for(i = 0; i < dora.length; i++){
				if(hai == dora[i]) doras += 4; //暗カン一つがドラなら4牌全てドラ
			}
		//暗カン内の赤ドラも探す
			for(int j = 0; j < 4; j ++){
				if(ankan_hai[i][j].hasAttribute(MJIHaiReader.ATTR_RED)) doras ++;
			}
		}

		//リーチした時のあがり点を概算する
		int han_suu = doras + 3;
		reached_agari_score = 30 << han_suu; //30符計算
		if(reached_agari_score >= 2000){ //満貫以上で計算する
			if(han_suu < 8)
				reached_agari_score = 2000; //満貫
			else if(han_suu < 10)
				reached_agari_score = 3000; //跳満
			else if(han_suu < 13)
				reached_agari_score = 4000; //倍満
			else if(han_suu < 15)
				reached_agari_score = 6000; //三倍満
			else reached_agari_score = 8000; //役満
		}
		if(iface.getCha() == 0) reached_agari_score *= 6; //親なら素点から6倍
		else reached_agari_score *= 4; //子なら素点から4倍
		}

//待ち牌の出現確率
double p = 1.0;
for(int i = 0; i < machihai_remain; i++){
		p += (66 - i) / (66 - 1 + tsumohai_remain);
}
p = 1.0 - p; //待ち牌の内、一枚が現れる確率
p /= 2.0; //リーチによる警戒を考え出現率を半減
```

```
		//リーチした時としない時で期待値を出し、一定以上望めるのであればリーチする
		int reach_score = (int)(((reached_agari_score - agari_score) / 100) * p);
		return reach_score >= 30;
	}

	//他家の捨て牌に対するリアクション
	public int onAction(int action, int player_no,int target_no,MJIHai hai){
		switch(action){
		case 512 : //リーチに対して
		case 256 : //捨て牌に対して
			if(player_no != 0){ //自分以外の捨て牌だったら鳴く
				return calcNaki(hai, player_no);
			}
		case MJPIR_ANKAN : //暗カン
		case MJPIR_MINKAN : //明カン
			if(player_no != 0){ //自分以外のカンだったらロンする
				int sel_ron = calcRon();
				if(sel_ron == 2)
					return MJPIR_RON;
			}
			return 0;
		}
		return 0;
	}


	//捨て牌を決めるロジック
	private int calc_sutehai(MJITehai tehai,MJIHaiReader tsumohai){
		int selected_hai = 0; //とりあえず捨て牌を仮に左端にセットする
		int score_max = -1; //最大評価値

		MJIHaiReader tehai_hai[] = tehai.getTehai();

		//ツモ牌の評価関数を計
		if(tsumohai != null){
			int tehai_score = eval_tehai(tehai_hai,tsumohai);
			int sutehai_score = eval_sutehai(tsumohai);

			score_max = tehai_score + sutehai_score;
			selected_hai = 13; //ツモ牌番号は13なのでとりあえずそれを最大評価値として返す
		}

		score_max = -1;
		selected_hai = 13;

		for(int i = 0; i < tehai_hai.length; i++){
			MJIHaiReader sutehai = tehai_hai[i]; // 捨て牌候補

			//捨て牌が一つ前の牌と同じであればスキップ
			if(i > 0) if (sutehai.equals(tehai_hai[i-1])) continue;
			//候補が捨てられない牌ならスキップ
			if(! iface.isHaiThrowable(sutehai.getHaiNo())) continue;
			//捨て牌とツモ牌を交換する。ツモ牌がない場合（鳴き後）はnull
			tehai_hai[i] = tsumohai;

			//捨てた後の手配の評価関数
			int tehai_score = eval_tehai(tehai_hai,sutehai);
			//捨てる牌自体の評価関数
			int sutehai_score = eval_sutehai(sutehai);
			//総合評価関数
			int score = tehai_score + sutehai_score;
```

```
                        if(score > score_max){
                                selected_hai = i;
                                score_max = score;
                        }
                                tehai_hai[i] = sutehai;
                }
                        return selected_hai;
}

//鳴くに関するクラス
private int calcNaki(MJIHaiReader hai,int player_no){
                MJITehaiReader tehai = iface.getTehai();

                //ロンをするか
                int sel_ron = calcRon();
                if(sel_ron == 2) //ロンの条件を満たす
                        return MJPIR_RON;
                if(sel_ron == 1) //テンパイ時
                        return 0
                //MJIHaiReader [] tehai_hai = tehai.getTehai();
                MJIHaiReader tehai_hai [] = tehai.getTehai();
                setTeCnt(tehai_hai);

                int hai_no = hai.getHaiNo();

boolean menzen = tehai.getMinkans().length + tehai.getMinkos().length + tehai.getMinshuns().length == 0;
//手牌に同一牌が2つ以上あってかつ字牌である
                if(menzen){
                        if(hai_no>= 27 && te_cnt[hai_no] == 2){
                                //字牌であってもオタ風であっては鳴かない
                                if(hai_no >= 31 || hai_no - 27 == iface.getCha() || hai_no - 27 == iface.getKyoku() / 4){
                                //ドラの時は鳴く
                                int[] doras = iface.getDora();
                                for(int i = doras.length - 1; i >= 0; i--){
                                        if(doras[i] == hai_no)    return MJPIR_PON;
                                }

                                //自分の点数が2位より8000点以上なら無条件でポンをする
                                int my_score = iface.getScore(0);
                                int max_sc_sa = 0;
                                int sc_sa = 0;
                                for(int i = 1; i < 4; i++){
                                        sc_sa = my_score - iface.getScore(i);
                                        if(sc_sa > max_sc_sa) max_sc_sa = sc_sa;
                                }
                                        if(sc_sa > 8000) return MJPIR_PON;
                                }
                        }
                        //残り山牌が12以下なら形式テンパイのための処理
                        //そうでない場合は以下をスルーする
                        if(iface.getHaiRemain() > 12)
                                return 0;
                        }
                        int max_score = eval_tehai(tehai_hai,null); //現在の手牌の評価点を読み込む
                        int ret = 0;

                        //既に副露している場合は価値のある牌だけを鳴く
                        if(te_cnt[hai_no] >= 2){
                                int sc = eval_tehai_in_naki(tehai,hai,hai_no,hai_no);

                        //ポンの場合
                                if(sc > max_score){
```

```
                                        max_score = sc;
                                        ret = MJPIR_PON;
                                }
                        }
                        //チーの場合
                        //字牌以外 or 上家以外からはチー出来ない
                        if(player_no == 3 && hai_no < 27){
                                int kazu = (hai_no % 9) + 1; //数牌の数の値
                                if(kazu < 8) //左端をチーする場合
                                        if(te_cnt[hai_no + 1] > 0 && te_cnt[hai_no + 2] > 0){
                                                int sc = eval_tehai_in_naki(tehai, hai, hai_no + 1, hai_no + 2);
                                                if(sc > max_score){
                                                        max_score = sc;
                                                        ret = MJPIR_CHII1;
                                                }
                                        }
                                if(kazu > 2)
                                        if(te_cnt[hai_no - 1] > 0 && te_cnt[hai_no - 2] > 0){
                                                int sc = eval_tehai_in_naki(tehai, hai, hai_no - 1, hai_no - 2);
                                                if(sc > max_score){
                                                        max_score = sc;
                                                        ret = MJPIR_CHII2;
                                                }
                                        }
                                if(kazu > 1 && kazu < 9)
                                        if(te_cnt[hai_no - 1] > 0 && te_cnt[hai_no + 1] > 0){
                                                int sc = eval_tehai_in_naki(tehai, hai, hai_no + 1, hai_no - 1);
                                                if(sc > max_score){
                                                        max_score = sc;
                                                        ret = MJPIR_CHII3;
                                                }
                                        }
                                }
                        return ret;
                }
        }
        //ロンするかの条件0ロンしない,1ダマテン,2ロンをする
        private int calcRon(){
        //テンパイしているか していれば何待ちかを取得
        boolean machi[] = new boolean[34];
        boolean b_tempai = iface.getMachi(machi);

        if(b_tempai){
                int agari_score = iface.getAgariScore(); //素の上がり点
                //フリテンかどうか 上がり点が0ならフリテン,捨て牌に待ち牌があればフリテン
                if(agari_score > 0){
                        for(int i = 0; i < 34; i++){
                                if(machi[i])
                                        if(iface.isHaiAnpai(0,i))
                                                return 1;
                        }
                        //親なら無条件で上がる
                        if(iface.getCha() == 0)
                                return 2;
                        //一つ上の順位との点差と残り局数を考慮する
                        int gap = 0; //一つ上の順位との点差
                        int rank = 0; //自分の順位
                        int my_score = iface.getScore(0); //自分の点数
                                for(int i = 0; i < 4; i++){
                                        int sa = iface.getScore(i) - my_score;
                                        if(sa > 0){
                                                rank ++;
```

```
                                        if(gap == 0 || sa < gap) gap = sa;
                                }
                        }
                //トップなら無条件で上がる
                if(rank == 0)
                        return 2;

                agari_score += iface.getHonba() * 300; //本場数掛ける300点を上がり点に付加する
                int get_score = agari_score + iface.getReachBou() * 1000;
                                        //供託リーチの数掛ける1000点を上がり点に付加する

                //オーラスを考慮する
                if(iface.getKyoku() == i_oras_kyoku){
                        int agari_go_score[] = new int[4];
                        for(int i = 0; i < 4; i++){
                                agari_go_score[i] = iface.getScore(i);
                        }
                        agari_go_score[0] += get_score;
                        agari_go_score[0] -= agari_score;
                        //上がった後の点数計算
                        int agari_go_rank = 0;
                        for(int i = 0; i < 4; i++){
                                if(agari_go_score[i] > agari_go_score[0]) agari_go_rank++;
                                }
                                if(agari_go_rank < rank)
                                        return 2;
                                } else {
                                int border = gap / (i_oras_kyoku - iface.getKyoku() + 1);
                                if(get_score >= border)
                                        return 2;
                                }
                        return 1;
                        }
                return 1;
                }
        return 0;
        }
}

int i_oras_kyoku; //オーラス時の特別な思考をするため

//通常時は3局終了で次局オーラス,西入なら7局終わりでオーラスと認識する
public void onStartGame(){
        if(iface.getRule(MIPIface.MJRL_NANNYU) == 0)
                i_oras_kyoku = 3;
        else if (iface.getRule(MIPIface.MJRL_SHANYU) == 0)
                i_oras_kyoku = 7;
        else i_oras_kyoku = 15;
}

//ゲームを引き続きする時の処理
public boolean onExchange(int state){
        MJITehaiReader tehai = iface.getTehai();
        tenpai_flag = iface.getMachi(tehai,machi);

        menzen = tehai.getMinshuns().length==0 && tehai.getMinkos().length==0 && tehai.getMinkans().length==0;
        nakiok_flag = !menzen;
        sthai = -1;
        tehai_score = eval_tehai_sub(false);
        return true;
}
```

```
		//ゲームを引き続きを可否の確認
		public boolean isExchangeable(){
			return true;
		}

		private int search(MJITehaiReader tehai,int hai){
			int i;
			MJIHaiReader[] hais = tehai.getTehai();
			int j = hais.length;
			for(i=0;i<j;i++) if (hais[i].getHaiNo() == hai) return i;
			return 13;
		}
}
```