卒業研究報告書

題目

配牌時の役満和了率を考慮した麻雀 AI の開発

指導教員

石水 隆　講師

報告者

13-1-037-0003

豊山 力登

近畿大学理工学部情報学科

平成 29 年 1 月 31 日提出

概要

麻雀は不完全情報ゲームであり，完全な解析は行えず必勝法は存在しない．しかし，自分の手牌や対戦相手の河牌などから，勝率を上げるための戦略を考えることはでき，麻雀の上級者は様々なメソッドに基づき打つべき手を判断している．

麻雀には色々な役があり，和了し易さや和了点が異なる．役の中でも特に役満とされる役は，和了できる確率は低いが和了できれば高得点を得ることができ，その後のゲームを優位に進めることができる．そこで本研究では，効率良く役満を和了するために，どのような配牌のときに役満を狙うべきかを検証する．

本研究では，役満を狙うべき条件を検証するために，麻雀 AI を作成し，AI 同士を対戦させて統計データを取る．役満には比較的和了し易いものから、和了率の極端に低い難易度の高いものまで様々ある．そこで本研究では、和了率が比較的高い国士無双，四暗刻，大三元に関する検証を中心に行う．

目次

1 序論..... 1

2 研究内容..... 2

3 麻雀 AI プログラム..... 5

3.1Kokusi.java クラス, Suuan.java クラス, Daisan,java クラス 共通部分.....5

3.1.1 変数..... 5

3.1.2 メソッド..... 5

3.2Kokusi.java クラス,..... 5

3.2.1 メソッド..... 6

3.3Suuan.java クラス..... 6

3.3.1 変数..... 6

3.3.2 メソッド..... 6

3.4Daisan.java クラス..... 7

3.4.1 変数..... 7

3.4.2 メソッド..... 7

4 実験結果..... 7

5 結論..... 9

参考文献..... 12

付録 付録 A..... 14

付録 付録 B..... 16

付録 付録 C..... 18

1　序論

1.1 本研究の背景

麻雀は多人数（３～４人）零和不完全情報ゲームであり，不確定性要素（配牌やツモ牌がランダム）が大きいためプレーヤー自身の運によって勝敗が大きく左右されるので最善手を判断するのが難しい．そのため麻雀 AI は他のオセロ・将棋・囲碁などに比べて開発が進んでいない．しかし，麻雀には必勝法は存在しないものの，自分の手牌や対戦相手の河牌などから，勝率を上げるための戦略を考えることはできる．麻雀の上級者とされる人々は様々なメソッドに基づき打つべき手を判断しており，麻雀の初心者よりも高い勝率を得ることができる．従って，麻雀 AI においても優れた戦略を組み込むことができれば，高い勝率となることが期待できる．

麻雀には色々な役があり，和了し易さや和了点が異なる．麻雀で勝利するためには，どの役を狙うかを決めることは重要であり，狙った役で和了するためにどの牌を切るかを決定しなければならないまた，麻雀では対戦相手に放銃してしまうと大量の得点を支払う必要があるので，負けないためには放銃しないための戦略も必要となる．

麻雀に限らず勝負事には，オカルトやジンクスと言われる科学的には証明できない運やツキを勝ち要素と考える流れ論，または精神論（例えば，野球なら球場との相性が悪いから負けた．勝ち運のついたパンツを履いていたから勝った．麻雀なら，流れが悪いので副露した．萬子を捨てると待ちが広いが，筒子を捨てると待ちが狭い状態で，筒子に好かれているので待ちの狭い筒子を捨てて立直をかけた次のツモ牌で和了した．など）と[5]，確率や証明など科学的根拠や事実を勝ち要素と考えるデジタル論の２つの論がある．[2][3]現在の麻雀では，デジタル論での戦術が勝利に繋がると考える人々も多いが，実際に流れ論を唱えるプロ雀士も多く存在し，流れ論派が勝利する事も珍しくはない．

麻雀には役満と言われる最高役が存在し，非常に和了率が低いが，もし和了できれば大量の得点を稼ぎつつ，他のプレイヤーに精神的なプレシャーを与えることができ，以降のゲームを有利に進め易くなる．しかし，役満は配牌時にある程度必要な牌が揃っていないと和了することはできない．

そこで本研究では，効率良く役満を和了するために，役満のみ（本研究では国士無双・四暗刻・大三元）を狙い続ける麻雀 AI を作成し，それを用いて配牌時の手牌毎の役満和了率を調査し，どのような配牌のときに役満を狙うべきかを検証する．

1.2 本研究の目的

麻雀は，一昔前までイメージの悪い博打のためのゲームと多くの人々に認知されていたが，現在では，数値と確率論を元にいかに効率良く和了するかが鍵となるデジタルな数値ゲームとなりつつある．また実際に，莫大な量のデータをもとに，回数が多くなればなるほど理論値に近づく大数の法則を用いてそれを証明する書籍なども出版されており，麻雀においての数値，確率のデータを用いた戦術が有効である事は確かである．[2][3][5]

そこで本研究では，麻雀の不確定性要素に着目し，役満と呼ばれる最高役を和了する事のできる確率が高い手牌を調査し，役満をより効率良く和了する事のできる手牌を検証する事を目的とする．

1.3 麻雀に関する既知の戦略

本節では，麻雀に関する既知の戦略について述べる．

麻雀では，現物切りと呼ばれる既に捨てられた牌を安牌として捨て牌にする戦略や筋切りと呼ばれる「1，4，7」「2，5，8」「3，6，9」でフリテンが予測される牌を安牌として捨て牌にする戦略.また，立直をかける事ができる状態でも他の役があれば，立直しなかったり（ダマテン），その場の点数差や場数・局数から狙うべき役を変える（etc...オーラスで１位になることができる役を狙う）戦略などが一般的に知られている[5][6].

麻雀は各役で和了し易さが異なり，同じ点数の役でも出やすい物と出にくいものがある．和了し易さについては，牌の種類と牌の個数から各役の出現率が求められている[4]．表１に，各役の出現率を

示す.

1.4 本報告書の構成

本報告書の構成は以下の通りである．まず２章で本研究で適応した麻雀のルールと研究内容を解説する．３章で本研究で作成した麻雀 AI について説明し，４章で麻雀 AI 同士の対戦による実験結果について述べる。最後に５章で結論および考察・今後の課題を述べる.

2　研究内容

本章では，麻雀のローカルルールを含めた研究中の適応ルールと具体的な研究内容を述べる.

2.1 麻雀のルール

麻雀は３～４人で行なわれる１３６枚あまりの牌を使ったテーブルゲームであるが，ローカルルールが存在するため，本研究で適応したルールを以下に示す.

- 麻雀は一般的に種類１３６枚の牌を使用する．牌には萬子，筒子，索子，字牌があり，萬子，筒子，索子は各１から９までの９種，字牌は白發中の三元牌と東南西北の四風牌から成る.
- プレイヤーにゲーム開始時に配られる点棒（点数の書かれた棒）を奪い合い，最終的に持っている点数の多さで順位が決まる.
- 各プレイヤーは手牌１３枚と和了牌１枚を合わせた１４枚の牌で定められた形を揃えることを目指す.
- 自分に配られた牌を組み合わせて役を作り，和了する事で他プレイヤーから点数を奪う事ができる.
- より高い役を作る事によって，より多くの点数を奪う事ができる.
- サイコロで親決めを行なった後，ゲームが開始される．ゲームの進行は局で管理される.
- １局は，牌が各プレーヤーに配られる配牌から，誰かが和了する，あるいは流局するまで続き，その局を一定数こなすか，プレーヤーの点数が無くなるまで続く.
- 現代の麻雀では，半荘と呼ばれる東場・南場の２つの場でゲームの局数が管理される．東場・南場，各々１～４局までで構成される．本研究でも半荘を１ゲームとしている.
- ４人打ち・・・・・基本的に麻雀は４人で行なわれるが，よりギャンブル性の高い３人麻雀も存在する．本研究では４人打ちを適応している.
- 南入・・・・・本研究では無条件で南入する.
- 西入・・・・・本研究ではいかなる場合でも西入しない.
- 親流れ・・・・・本研究では東場，南場ともに親がノーテンで親流れとする.
- ドラ・・・・・本研究では裏ドラあり，カン裏ドラあり，赤ドラ無しとする.
- リーチ後の暗槓・・・・・本研究ではリーチ後の暗槓を認める.
- トリプルロン・・・・・本研究では重複ロンを認める.
- ２翻縛り・・・・・本研究では２翻以上からの和了のみを認める.
- ピンヅモ・・・・・ピンフという和了役は本来はロン和了時のみ発生する役だが，本研究ではツモ和了時にも発生するものとする.
- 喰いタン・・・・・本研究では福露している状態での断公九も１翻と数える.
- カラテン・・・・・本研究ではテンパイ時に待ち牌が全て河に捨てられており和了できない状態でもテンパイ扱いとする.

2.2 麻雀の役

本節では、麻雀の役について述べる.

以下に麻雀の役を列挙する.

- 立直・・・・・門前で聴牌した状態で捨牌するとき，リーチ宣言し１０００点の供託料を払い，上がったときに成立する.
- 役牌・・・・・白・發・中・門風牌（自風牌．親なら東）・荘風牌（場風牌．親なら東）の刻子（槓子）がある場合に成立する.
- 断幺九・・・・・２～８の数牌のみの場合に成立する.
- 平和・・・・・メンツが４つとも順子で，雀頭が役牌でなく，両面待ちの時に成立する.
- 門前清模和・・・・・門前でツモアガリした場合に成立する.
- 一発・・・・・リーチ宣言をした後，一巡以内に上がった場合に成立する．ポン・チー・カン（暗カンを含む）が行われると無効になる.
- 混一色・・・・・万子，索子，筒子のどれか一種類の牌と，字牌だけの場合に成立する.
- 一盃口・・・・・同じ順子が２組ある場合に成立する.
- 三色同順・・・・・３種類の色（万子・索子・筒子）それぞれに，同じ数字の並びの順子がある場合に成立する.
- 対々和・・・・・４つのメンツすべてが刻子（槓子）の場合に成立する.
- 七対子・・・・・対子が７組ある場合に成立する．同牌が４枚ある場合には成立しない.
- 一気通貫・・・・・同種（同色）の数牌で，１２３，４５６，７８９，の順子がある場合に成立する.
- 全帯・・・・・４つのメンツと雀頭すべてにヤオ九牌（1・9と字牌）が含まれている場合に成立する.
- 海底・・・・・局最後のツモ（海底）された牌によりツモアガリした場合．もしくは，局最後のツモ（海底）された牌が捨てられたときに，その捨牌でロンアガリした場合に成立する.
- 清一色・・・・・万子，索子，筒子のどれか一種類の牌だけの場合に成立する.
- 三暗刻・・・・・暗刻（暗槓）が３つある場合に成立する．ロンの場合，アガリ牌を含む刻子は暗刻とみなされないことに注意が必要である.
- 純全帯・・・・・４つのメンツと雀頭すべてに老頭牌（１・９）が含まれる場合に成立する.
- 嶺上開花・・・・・カンを行ったときに（不足する牌を補充するため）ツモってくる嶺上牌でツモアガリしたときに成立する．門前なら門前清模和とも複合する.
- 二盃口・・・・・同じ順子が２組という組み合わせが２つある場合に成立する.
- 小三元・・・・・三元牌（白・發・中）のいずれか１つを雀頭とし．残り２種類を刻子（槓子）とした場合に成立する.
- 混老頭・・・・・すべての牌がヤオ九牌（1・9と字牌）だけの場合に成立する.
- ダブル立直・・・・・自分の最初の捨牌でリーチ宣言して，上がったときに成立する.
- 四暗刻・・・・・暗刻（暗槓）が４つある場合に成立する．ロンの場合，アガリ牌を含む刻子は暗刻とみなされないことに注意が必要である.
- 三色同刻・・・・・３種類の色（万子・索子・筒子）それぞれに，同じ数字の刻子（槓子）がある場合に成立する.
- 大三元・・・・・三元牌（白・發・中）の３種類すべてが刻子（槓子）である場合に成立する.
- 国士無双・・・・・13種類すべてのヤオ九牌（１・９と字牌）が最低１枚ずつあり，そのうちのどれか１種類が２枚ある場合に成立する.
- 四喜和・・・・・風牌（東・南・西・北）のうち３種が刻子（槓子）で，残りの１種が雀頭か刻子（槓子）である場合に成立する.

- 字一色・・・・・すべての牌が字牌である場合に成立する．
- 清老頭・・・・・すべての牌が老頭牌（１・９）だけである場合に成立する．
- 地和・・・・・子の最初のツモでツモアガリした場合に成立する．そのツモの前にポン・チー・カン（暗カンを含む）が行われると無効になる．
- 緑一色・・・・・索子の２３４６８と字牌の發のみである場合に成立する．
- 九連宝燈・・・・・万子，索子，筒子のどれか一種類の牌だけで，１１１２３４５６７８９９９の牌とさらに１枚１～９の牌を追加した牌形である場合に成立する．
- 四槓子・・・・・槓子（暗槓・明槓いずれでも可）が４つある場合に成立する．
- 天和・・・・・親の配牌の時点で既にアガリ形になっている場合に成立する．

2.3 研究内容

本研究では，参考資料[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．本研究では表 1 で示したように，確率上役満で最も和了し易いとされる国士無双・四暗刻・大三元のみを狙い続ける麻雀 AI を作成し，AI 同士を対戦させて様々な手牌に対する役満の和了率を求める．配牌時の手牌がどのような時に役満を狙うべきかを検証するため，配牌時の手牌を国士無双なら么九牌，四暗刻なら暗刻と対子の数，大三元なら三元牌の数をそれぞれ増やして，各々 30 万局の対戦データから和了率を算出し，どのような手牌の時に役満を狙うべきかを検証する．

表 1 各役の出現率[4]

役	出現確率[%]		役	出現確率[%]
立直	４１．６		嶺上開花	０．３
役牌	３３．０		二盃口	０．１
断么九	２２．４		小三元	０．１
平和	２１．５		混老頭	０．１
門前清自摸和	１８．８		ダブル立直	０．１
一発	１０．２		四暗刻	０．０４
混一色	６．３		三色同刻	０．０４
一盃口	４．６		大三元	０．０３
三色同順	３．８		国士無双	０．０３
対々和	３．２		四喜和	０．０１１
七対子	２．４		字一色	０．００５
一気通貫	１．８		清老頭	０．００１８
混純全帯	１．０		地和	０．００１５
海底	０．９		緑一色	０．００１１
清一色	０．８		九連宝燈	０．０００４
三暗刻	０．７		四槓子	０．０００２
純全帯么九	０．４		天和	０．００００.....

3　麻雀 AI プログラム

本章では，本研究で作成した麻雀 AI プログラムについて説明する．

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．本研究で作成した麻雀 AI は，それぞれ国士無双狙い，四暗刻狙い，大三元狙いをするためのクラス Kokusi.java, Suuan.java,Daisan.java を持つ．以下でそれぞれのクラスについて述べる．

3.1 Kokusi.java クラス, Suuan.java クラス, Daisan,java クラス 共通部分

本節では，３つのクラスに共通している部分について述べる．

3.1.1　変数

以下に共通する変数を列挙する．

- iface 変数

iface 変数は，アクセス修飾子 private，MIPIface 型のインスタンス変数である．
この変数を用いて，メソッドを呼び出す事で様々なアクションを実行している（捨て牌を選択する．手牌を取得する．副露する．和了する．など）．
MIPIface クラスのインスタンスを格納している．

3.1.2　メソッド

以下に共通するメソッドを列挙する．

- intialize(MIPIface i)メソッド

intialize()メソっドは，アクセス修飾子 public，戻り型 boolean 型の MIPIface クラスのインスタンスを初期化するメソッドである．渡された MIPIface インスタンスをインスタンス変数 iface に保存している．

- getName() メソッド

getName()メソッドは，アクセス修飾子 public，戻り型 String 型の，AI の名前を返すメソッドである．

- onAction(int action, int player_no, int target_no, MJIHaiReader hai)メソッド

onAction()メソッドは，アクセス修飾子 public，戻り型 int 型の，立直，ツモ，ロン，カン，ポン，チーのいずれかのアクションを起こす選択を行なうメソッドである．
変数 action は起こしたアクションを表す変数である．
変数 player_no はアクションを起こしたプレイヤーを表す変数である．
変数 target_no はアクションの対象となったプレイヤーを表す変数である．
変数 hai はそのアクションに関連する牌を表す変数である．
今回は役満のみを狙うのでツモとロンの条件を和了点３２０００点以上としている．

3.2　Kokusi.java クラス,

Kokusi.java は国士無双だけを狙うクラスである．以下に Kokusi.java の変数とメソッドの説明を行

なう．

3.2.1　メソッド

以下に Kokusi.java 独自のメソッドを列挙する．

- onSutehai(MJITehaiReader te, MJIHaiReader tsumohai)メソッド

onStehai()メソッドは，アクセス修飾子 public，戻り型 int 型の捨て牌を選択するメソッドである．
変数 hai[]は手牌を格納した配列である．
変数 te は自分の手牌を表す変数である．
変数 tsumohai はツモってきた牌を表す変数である．
このメソッドはまず，ツモ牌で和了できるか（門前清自摸和）を判定する．
国士無双のみを狙うクラスであるから，まず么九牌以外の牌を全て捨てる．
么九牌のみから構成される手牌になったら，暗刻になっている牌を捨てる．
和了するためには頭（同じ牌が２つの組み合わせ）が必要なので，手牌から暗刻が無くなったら，最初の対子のみを保存しそれ以降の対子になってる牌を捨てる．
これを繰り返し行ない，テンパイの状態になれば和了牌以外はツモ切りする．

3.3 Suuan.java クラス

Suuan.java は四暗刻だけを狙うクラスである．以下に Suuan.java の変数とメソッドの説明を行なう．

3.3.1　変数

以下に Suuan.java 独自の変数を列挙する．

- kawa 変数（配列）

河に捨てられた牌の格納を行なう変数である．

- kawa1 変数（配列）

下家の河牌を格納する変数である．

- kawa2 変数（配列）

対面の河牌を格納する変数である．

- kawa3 変数（配列）

上家の河牌を格納する変数である．

- th 変数（配列）

手牌を格納する変数である．

3.3.2　メソッド

以下に Suuan.java 独自のメソッドを列挙する．

- onSutehai(MJITehaiReader te, MJIHaiReader tsumohai)メソッド

onStehai()メソッドは，アクセス修飾子 public，戻り型 int 型の捨て牌を選択するメソッドである．
変数 hai[]は手牌を格納した配列である．
変数 te は自分の手牌を表す変数である．
変数 tsumohai はツモってきた牌を表す変数である．

このメソッドはまず，ツモ牌で和了できるか（門前清自摸和）を判定する．
四暗刻のみを狙うクラスであるから，まず刻子と対子になっている牌以外を捨てる．
このとき槓子になっている牌も捨てる仕様となっている．役満狙いにおいて，槓をするメリットがほとんど無いためである．
手牌が暗刻と対子のみになったら，対子の中で河牌に既に２枚以上捨てられている牌を捨てる．刻子にすることができないからである．
これを繰り返し行ない，テンパイの状態になれば和了牌以外はツモ切りする．

3.4 Daisan.java クラス

Daisan.java は大三元だけを狙うクラスである．以下に Daisan.java の変数とメソッドの説明を行なう．

3.4.1　変数

以下に Daisan.java 独自の変数を列挙する．

- th 変数（配列）

手牌を格納する変数である．

3.4.2　メソッド

以下に Daisan.java 独自のメソッドを列挙する．

- onAction(int action, int player_no, int target_no, MJIHaiReader hai)メソッド

onAction()メソッドは，大三元狙いの場合のみ，三元牌を副露する仕様になっている．
その他は，他のクラスと共通する．

- onSutehai(MJITehaiReader te, MJIHaiReader tsumohai)メソッド

onStehai()メソッドは，アクセス修飾子 public，戻り型 int 型の捨て牌を選択するメソッドである．
変数 hai[]は手牌を格納した配列である．
変数 te は自分の手牌を表す変数である．
変数 tsumohai はツモってきた牌を表す変数である．
このメソッドはまず，ツモ牌で和了できるか（門前清自摸和）を判定する．
大三元のみを狙うクラスであるから，三元牌以外を捨てる．この時，三元牌以外の牌でも，手牌に２枚上ある牌や連番になっている牌は捨てない（従って，三元牌以外で牌が１つしかないものを捨てる）．
更に，対子の中で河牌に既に２枚以上捨てられている牌を捨てる．
これを繰り返し行い，テンパイの状態になれば和了牌以外はツモ切りする．

4　実験結果

本章では麻雀 AI 同士の対戦を行う計算機実験の結果を述べる．

4.1 国士無双狙いの内容

本節では，国士無双狙いの結果について述べる．参考資料[1]の既存の AI ３体と Kokusi.java １体の合計４体で，各条件３０万局ずつ統計をとった．配牌時の么九牌の個数が９個，１０個，１１個の時の国士無双の和了率を検証した．その結果を表２に示す．ただし，配牌時に同じ種類の么九牌はなく，全て違う種類の么九牌としている．

表２より，么九牌１１個時に積み込み無し時の１００倍以上の確率が算出された．従って，配牌時に么九牌１１個以上であれば国士無双を狙うべきであると言えが，配牌時に么九牌が１１個も揃うことはまずありえずない．一方、９個・１０個の場合は配牌時に揃っている事が頻繁に起こりえる．この事から，配牌時に頻繁に揃い得る個数かつ，積み込み無し時の１０倍以上の和了率が算出された１０個の場合以上の么九牌が配牌時に手牌に揃っていれば，国士無双を狙う価値が十分にあると考えられる．

表２ 国士無双の和了率 (試行回数 30 万回)

積み込み数	和了回数	割合
積み込み無し	３０万局中２１０回	０．０７％
么九牌９個	３０万局中１２９０回	０．４３％
么九牌１０個	３０万局中２４９０回	０．８３％
么九牌１１個	３０万局中２６９１０回	８．９７％

4.2 四暗刻狙いの結果

本節では，四暗刻狙いの結果について述べる．参考資料[1]の既存の AI ３体と Kokusi.java １体の合計４体で，各条件３０万局ずつ統計をとった．配牌時の暗刻の個数が１の時，対子無し，対子１，対子２，対子３，対子４．暗刻の数が２個の時，対子無し，対子１，対子２．暗刻の数が３個の時，対子無し，対子１のそれぞれの場合の和了率を検証した．その結果を表３に示す．

表３より，暗刻３，対子１時に積み込み無し時の２０倍以上の確率が算出された．従って，配牌時に暗刻３，対子１個以上であれば四暗刻を狙うべきであると言えが，配牌時にそれだけ揃う確率は現実的な値ではない．しかしながら，四暗刻が他の２つの役満に比べて積み込み無し時の和了率が極端に高い事と，四暗刻が役満であることのメリットを考えると，０．３％前後の和了率でも十分に狙う価値があるので，暗刻・対子が合わせて２個以上あれば狙っても良いと考えられる．

表３ 四暗刻の和了率 (試行回数 30 万回)

積み込み数	和了回数	割合
積み込み無し	３０万局中３８７回	０．１２９％
暗刻１	３０万局中５７０回	０．１９％
暗刻１，対子１	３０万局中９３０回	０．３１％
暗刻１，対子２	３０万局中８７４回	０．２９１３％
暗刻１，対子３	３０万局中１７７２回	０．５９０６％
暗刻１，対子４	３０万局中１８１２回	０．６０４％

暗刻２	３０万局中１０８９回	０．３６３％
暗刻２，対子１	３０万局中２１６５回	０．７２１６％
暗刻２，対子２	３０万局中３０６１回	１．０２０３％
暗刻３	３０万局中５４９０回	１．８３％
暗刻３，対子１	３０万局中８６７６回	２．８９２％

4.3 大三元狙いの結果

本節では，大三元狙いの結果について述べる．参考資料[1]の既存の AI ３体と Kokusi.java １体の合計４体で，各条件３０万局ずつ統計をとった．配牌時の三元牌の数がそれぞれ白，發，中の順番で「１，１，１」「２，１，１」「２，２，１」「２，２，２」「３，１，１」「３，２，１」「３，２，２」「３，３，２」「３，３，３」とし，それぞれの場合の和了率を検証した．その結果を表４に示す．

表４より，白３，發３，中２時に積み込み無し時の１００倍以上の確率が算出されたが，配牌時にそれだけ揃う確率は現実的な値ではない．そこで，表４を注意深く見てみると，三元牌の刻子・対子の個数に関わらず，三元牌が５個以上であれば積み込み無し時のほぼ８倍以上の和了率が算出されている．この事から，配牌時に三元牌が５個以上あれば大三元を狙うべきであると言える．

表４ 大三元の和了率 (試行回数 30 万回)

積み込み数	和了回数	割合
積み込み無し	３０万局中１８９回	０．０６３％
白１，發１，中１	３０万局中２７４回	０．０９０６％
白２，發１，中１	３０万局中１４４６回	０．４８２％
白２，發２，中１	３０万局中６３００回	２．１％
白２，發２，中２	３０万局中９６９１回	３．２３０３％
白３，發１，中１	３０万局中１９１６回	０．６３８６％
白３，發２，中１	３０万局中９６３３回	３．２１１％
白３，發２，中２	３０万局中１７６７０回	５．８９％
白３，發３，中２	３０万局中２２９８８回	７．６６％

5　結論

5.1 結論および考察

本研究では配牌時の手牌から役満和了率を計算し，役満を狙う AI を作成した．研究結果で示した通り，国士無双は么九牌が１１個以上の時に約９％の和了率があるので１１個あれば国士無双を狙う価値があると考えられる．

しかし，配牌時に手牌が揃う現実的な数値を考慮し，么九牌が１０個の時でも積み込み無し時の１０倍以上の確率があること，役満を和了することのメリットを考えると么九牌が１０個以上あれば，国士無双を狙う価値があると思われる．

四暗刻は，暗刻が配牌時に３つあっても積み込み無し時の１４倍，暗刻３つ対子１つの時でも約

３％と国士無双に比べると極端に和了率は上がらなかった．

しかし，配牌時に手牌が揃う現実的な数値を考慮し，表１で示した他の役満の和了率を考えると０３％前後でも十分だと考えられるため，配牌時に暗刻・対子が合わせて２個以上あれば四暗刻を狙う価値があると考えられる．

また，四暗刻は本研究で検証した役満の中で最も積み込み無し時の和了率が高かった．これは牌の制約がないため（どの牌を使っても良い）に，途中で手を変えたり（中が３枚切れれば中を暗刻にも頭にもできないので捨てる），相手に狙いがばれにくい（国士無双や大三元は，么九牌以外を切ったり三元牌を副露したりするので警戒されやすい）などの要因が考えられる．この事から，配牌が役満和了に向けて好ましくない場合は四暗刻を狙うのが良いと考えられる．

大三元は本研究の役満の中で唯一副露が許された役満である．そのため三元牌を対子で持っている場合でも和了率が格段に上がる．和了率で考えると三元牌が暗刻と対子関係なく合わせて５つ以上あれば表１の七対子や対々和とほぼ同等もしくはそれ以上の和了率が見込まれるため，三元牌が暗刻と対子関係なく合わせて５つ以上あれば十分に大三元を狙う価値があると考えられる

5.2 今後の課題

今回は，国士無双・四暗刻・大三元のみを研究対象としたが，今後の課題として他の役満の和了率，役満以外の手の和了率を計算し思考ルーチンの開発に役立てる事が必要だと思われる．

また，本研究では役満を狙う途中に役満以外の手で和了できても和了していない（大三元中に小三元で和了できてもしない．など）．勝率を上げる事を考えるならば，役満からの手変えのし易さなども考慮しながらAIを開発する事が必要になると考えられる．

さらに，今回の役満和了率を考慮したAIを実際に開発し勝率データをとり，この研究の意義を証明されたい．

謝辞

本研究を行なうにあたり，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 著：コンピュータ麻雀のアルゴリズム, IO-Books, 工学社(2007).

[2] とつげき東北 著, 福地誠 編：おしえて！科学する麻雀, 洋泉社(2009).

[3] とつげき東北 著：科学する麻雀, 講談社現代新書(2004).

[4] 星, 麻雀役の一覧(出現確率), 麻雀％ (2016), http://mjclv.com/yaku/syutugenn.html

[5] 麻雀ルールと役の解説，初心者のための麻雀講座 (2012),http://www2.odn.ne.jp/cbm15900/html/k14-1.html

[6] 浅尾豪, 大西建輔, 複数クライアントによる麻雀の協調支援手法の提案, ゲームプログラミングワークショップ 2015 論文集，Vol.2015, pp.88-91, 情報処理学会，(2015), http://id.nii.ac.jp/1001/00145756/

[7] 築地毅, 柴原一友, ディープラーニング麻雀 －オートエンコーダとドロップアウトの有効性－, ゲームプログラミングワークショップ2015 論文集, Vol.2015, pp.136-142, 情報処理学会, (2015), http://id.nii.ac.jp/1001/00145764/

[8] 海津純平, 成澤和志, 篠原歩, 一人麻雀における打ち方を考慮した評価指標に関する研究, ゲームプログラミングワークショップ 2015 論文集, Vol.2015, pp.172-178, 情報処理学会, (2015), http://id.nii.ac.jp/1001/00145770/

[9] 水上直紀, 鶴岡慶雅, 期待最終順位に基づくコンピュータ麻雀プレイヤの構築, ゲームプログラミングワークショップ 2015 論文集，Vol.2015, pp.179-186, 情報処理学会，(2015), http://id.nii.ac.jp/1001/00145771/

[10] 田中悠, 池田心, 麻雀初級者のための状況に応じた着手モデル選択, 研究報告ゲーム情報学 (GI), Vol.2014-GI-31, No.10, pp.1-8, 情報処理学会，(2014), http://id.nii.ac.jp/1001/00099267/

[11] 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, No.11, pp.1-6, 情報処理学会，(2014), http://id.nii.ac.jp/1001/00099268/

[12] 我妻敦, 原田将旗, 森田一, 古宮嘉那子, 小谷善行, SVR を用いた麻雀における捨て牌の危険度の推定，研究報告ゲーム情報学 (GI), Vol2014-GI-31, No.12,pp.1-6, (2014), http://id.nii.ac.jp/1001/00099269/

[13] 原田将旗, 古宮嘉那子, 小谷善行, 麻雀における手牌と残り牌からの上がり探索による着手決定アルゴリズム CHE, 研究報告ゲーム情報学(GI), Vol.2014-GI-31, No.13, pp.1-4, 情報処理学会, (2014), http://id.nii.ac.jp/1001/00099270/

[14] 水上直紀, 中張遼太郎, 浦晃, 三輪誠, 鶴岡慶雅, 近山隆, 降りるべき局面の認識による 1 人麻雀プレイヤの 4 人麻雀への適用, ゲームプログラミングワークショップ 2013 論文集, Vol.2013, pp.1-7, 情報処理学会, (2013), http://id.nii.ac.jp/1001/00095782/

[15] 中張遼太郎, 水上直紀, 浦晃, 三輪誠, 鶴岡慶雅, 近山隆, LinUCB の 1 人麻雀への適用, ゲームプログラミングワークショップ 2013 論文集, Vol.2013, pp.114-117, 情報処理学会 (2013), http://id.nii.ac.jp/1001/00095800/

[16] 根本佳典, 古宮嘉那子 , 小谷善行, CRF を用いた麻雀の不完全情報推定, ゲームプログラミングワークショップ 2012 論文集，Vol.2012, No.6, pp.155-158, 情報処理学会 (2012), http://id.nii.ac.jp/1001/00091346/

[17] 三木理斗 , 三輪誠 , 近山隆, 木カーネルを用いた SVM による麻雀打ち手の順位学習, ゲームプログラミングワークショップ 2008 論文集, Vol.2008, No.11, pp.60-66 情報処理学会 (2008), http://id.nii.ac.jp/1001/00097668/

[18] 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, 研究報告ゲーム情報学(GI), Vol.2000-GI-003, pp.65-70, 情報処理学会, (2000), http://id.nii.ac.jp/1001/00058644/

付録 国士無双狙いのプログラム

本研究で作成した Java プログラムのソースを以下に示す.

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

public class Kokusi extends MJ_AI {
        private MIPIface iface;

    public boolean initialize(MIPIface i)//インスタンス初期化
        {
                iface = i;
        return true;
        }


    public String getName() {//AI の名前を表示

                return "koku";
        }


    public int onSutehai(MJITehaiReader te,
                        MJIHaiReader tsumohai) {//捨て牌を選択
        MJIHaiReader hai[] = te.getTehai();//手牌配列

                if (iface.getAgariScore() > 32000){
                        return MJPIR_TSUMO; //ツモ

                }

                   for(int i = 0; i < hai.length; i++){
                int hai_no = hai[i].getHaiNo();

              int kazu = (hai_no % 9)+1;
              if(hai_no >= 25 && kazu != 1 && kazu != 9){
                    return MJPIR_SUTEHAI | i; //   やおちゅー牌以外
                    }}

             for(int i = 0; i < 11; i ++){
                int hai_no = hai[i].getHaiNo();
                int hai_no1 = hai[i+2].getHaiNo();

                 if(hai_no == hai_no1){
                    return MJPIR_SUTEHAI | i;//暗刻

                 }}
```

```
            for(int i = 0; i < 12; i ++){
                int hai_no = hai[i].getHaiNo();
                int hai_no1 = hai[i+1].getHaiNo();

                if(hai_no == hai_no1){//最初のトイツ

                  for(int j = i+2; j < 12; j++){
                        int hai_no2 = hai[j].getHaiNo();
                        int hai_no3 = hai[j+1].getHaiNo();
                        if(hai_no2 == hai_no3){// ２つめのトイツ
                            return MJPIR_SUTEHAI | j;//トイツ切る
                        }
                  }
                }
                else{
                   return MJPIR_SUTEHAI | 13;//つもぎり

                  }
              }
           return MJPIR_SUTEHAI | 13;//つもぎり
}

    public int onAction(int action, int player_no, int target_no,
                                    MJIHaiReader hai){//アクションを選択
          if (action == MJPIR_SUTEHAI || action == MJPIR_REACH) {
             if (iface.getAgariScore() > 32000)
                return MJPIR_RON;//ロン
          }
          return 0;


}
}
```

付録 四暗刻狙いのプログラム

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

public class Suuan extends MJ_AI {
   private MIPIface iface;

   public boolean initialize(MIPIface i)//インスタンス初期化
         {
                  iface = i;
                  return true;

}
   public String getName() {//AI の名前を表示
                  return "suuan";
         }


   public int onSutehai(MJITehaiReader te,
                              MJIHaiReader tsumohai) {//捨て牌を選択
            MJIHaiReader hai[] = te.getTehai();
            if (iface.getAgariScore() > 32000){
                              return MJPIR_TSUMO; //ツモ

                  }

                              int[] kawa;
            kawa = new int[34];//河牌配列
            for(int i = 0; i < kawa.length; i ++){
                  kawa[i] = 0;//初期化
            }

            MJIKawahaiReader kawa1[] = iface.getKawa(1);//下家の河牌
            MJIKawahaiReader kawa2[] = iface.getKawa(2);//対面の河牌
            MJIKawahaiReader kawa3[] = iface.getKawa(3);//上家の河牌

            for(int i = 0; i < kawa1.length; i ++){
                  int hai_no = kawa1[i].getHai().getHaiNo();
                  kawa[hai_no]++;//牌の配列 no に 1 プラス
            }
            for(int i = 0; i < kawa2.length; i ++){
                  int hai_no = kawa2[i].getHai().getHaiNo();
                  kawa[hai_no]++;//牌の配列 no に 1 プラス
            }
            for(int i = 0; i < kawa3.length; i ++){
                  int hai_no = kawa3[i].getHai().getHaiNo();
                  kawa[hai_no]++;//牌の配列 no に 1 プラス
```

```
                }

            int[] th;
            th = new int[34];//手配配列
            for(int i = 0; i < th.length; i ++){
                th[i] = 0;//初期化
                }

              for(int i = 0; i < hai.length; i ++){
                int hai_no = hai[i].getHaiNo();
                th[hai_no]++;//牌の配列 no に 1 プラス
                }


                 for(int i = 0; i < 13; i++){
                 int hai_no = hai[i].getHaiNo();
                 if(th[hai_no] == 1 || th[hai_no] == 4){
                    return MJPIR_SUTEHAI | i;//刻子、といつ以外
                 }
                 }
                 for(int i = 0; i < 13; i++){
                    int hai_no = hai[i].getHaiNo();
                    if(th[hai_no] == 2 && kawa[hai_no] >= 2){
                    return MJPIR_SUTEHAI | i;
                    }else if(th[hai_no] == 2){
                            return MJPIR_SUTEHAI | i;
                            }else{
                    return MJPIR_SUTEHAI | 13;//つもぎり
                 }
                 }




            return MJPIR_SUTEHAI | 13;//つもぎり
        }

    public int onAction(int action, int player_no,
                            int target_no, MJIHaiReader hai){//アクションを選択
        if (action == MJPIR_SUTEHAI || action == MJPIR_REACH) {
            if (iface.getAgariScore() > 32000)
                return MJPIR_RON;//ロン
        }
        return 0;
 }
}
```

付録 大三元狙いのプログラム

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

public class Daisan  extends MJ_AI {
   private MIPIface iface;

        public boolean initialize(MIPIface i)//インスタンス初期化
        {
                iface = i;
                return true;

}
        public String getName() {//AI の名前を表示
                return "daisan";
        }

   public int onSutehai(MJITehaiReader te,
                        MJIHaiReader tsumohai) {//捨て牌を選択

      MJIHaiReader hai[] = te.getTehai();

      if (iface.getAgariScore() >= 32000){
                        return MJPIR_TSUMO; //ツモ

            }
     int[] th;
      th = new int[34];//手配配列
      for(int i = 0; i < th.length; i ++){
            th[i] = 0;//初期化
            }

       for(int i = 0; i < hai.length; i ++){
            int hai_no = hai[i].getHaiNo();
            th[hai_no]++;//牌の配列 no に 1 プラス
            }


       for(int i = 0; i < (hai.length-1); i++){
             int hai_no = hai[i].getHaiNo();
             if(hai_no <= 30 && th[hai_no] == 1){
                return MJPIR_SUTEHAI | i;//三元牌以外
             }
             }
             for(int i = 0; i < (hai.length-1); i++){
                int hai_no = hai[i].getHaiNo();
                if(hai_no <= 30 && th[hai_no] == 2){
```

```
                    return MJPIR_SUTEHAI | i;
                }else{
                    return MJPIR_SUTEHAI | 13;//つもぎり
                }
                }

        return MJPIR_SUTEHAI | 13;



}

  public int onAction(int action, int player_no,
                            int target_no, MJIHaiReader hai){//アクションを選択


        if (action == MJPIR_SUTEHAI || action == MJPIR_REACH) {
            int hai_no = hai.getHaiNo();

             int[] th;
            th = new int[34];//手配配列
            for(int i = 0; i < th.length; i ++){
                    th[i] = 0;//初期化
                    }

              for(int i = 0; i < hai.length; i ++){
                    int hai_no = hai[i].getHaiNo();
                    th[hai_no]++;//牌の配列 no に 1 プラス
                    }

            if(hai_no >= 31 && th[hai_no] == 2){
                    return MJPIR_PON;//三元牌を副露
          }

            else if (iface.getAgariScore() >= 32000)
                    return MJPIR_RON;//ロン
        }
        return 0;

  }
}
```