卒業研究報告書

題目

Java を用いた京都将棋アプリの開発

指導教員

石水　隆　講師

報告者

15-1-037-0140

野々下 魁

近畿大学理工学部情報学科

平成 31 年 1 月 31 日提出

概要

京都将棋は盤面縦横 5 マスの盤面を用いるミニ将棋の一種である.京都将棋の最大の特徴は「香とと金」,「銀と角」,「金と桂」,「飛と歩」と王以外の駒の裏表に異なる駒が書かれており,駒を動かすとその駒を裏返すという特別なルールにより一手ごとに駒の性能が変わる点である.

一手ごとに駒の性能が変わるという奥深い将棋であるが、知名度が低くあまり研究されていない.そこで本研究では京都将棋のアプリケーションを作成し,強い AI を開発する.

将棋の AI を作成する場合，局面の評価値を求める要素として，各駒に評価値を設定する方法がよく用いられる.本将棋ではプロ棋士により,駒の評価値はほぼ決まっているため,それを用いることができる．一方，京都将棋は本将棋と裏表が異なり，また，盤面も５×５と小さいため，本将棋の駒の評価値をそのまま使うことはできない．そこで，本研究では,京都将棋において最適となる駒の評価値を検証する.

本研究では，Java を用いて京都将棋 AI を作成し，各駒に割り当てられた評価値が異なる AI 同士を対戦させ，最適な駒の評価値を求める.

目次

1. 序論

1.1 二人零和有限確定完全情報ゲーム

将棋や囲碁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.二人零和有限確定完全情報ゲームとは,二人とは人数が二人または,二つのグループによって ゲームを行うことである.零和とは、複数の人が相互に影響し合う状況の中で,全員の利得が常に零になること.有限とは,各プレイヤーの手番の組み合わせが必ず有限で終了することである.確定は,サイコロのようなランダム要素が存在しないことである.完全情報とは,互いのプレイヤーが相手の全ての情報が公開されていることである.

二人零和有限確定完全情報ゲームの特徴は,理論上は完全な先読みが可能であり,双方のプレイヤーが最善手を打った場合,先手必勝,後手必勝,引き分けかが必ず決まる点である.しかし多くのボードゲームでは選択する局面が極めて膨大であるため,完全解析が不可能である.例を挙げれば,総局面数はオセロなら 10^{28} 通り,チェスなら 10^{47} 通り,将棋なら 10^{69} 通り,囲碁なら 10^{169} 通り[1]あるとされており現在の計算機では計算不可能である.一方,局面数が少ないため完全解析されたゲームも複数存在する.例えば,五目並べに様々なルール変更を加えた連珠は,双方が最善手を打った時,47 手で先手が勝利する[2].はさみ将棋は双方が最善手を指せば千日手のような状況になり引き分けとなる.同様に,チェッカーも引き分けとなる[3].

局面数が大きいゲームについては,ゲーム盤をより小さいサイズに限定した場合の解析も行われている.サイズ 6×6 のリバーシでは,双方最善手を打つと 16 対 20 で後手勝ちとなる[4].また,サイズ 4×4 の囲碁は双方最善手を打つと引き分け[5],5×5 の囲碁は黒の 25 目勝ちとなる[6].将棋については,盤面のサイズや使用する駒の種類を減らしたサイズ 5 五将棋[7]やゴロゴロ将棋[8]などのミニ将棋がある.5 五将棋はサイズ 5×5 の盤と 6 種類の駒,ゴロゴロ将棋はサイズ 5×6 の盤と 4 種類の駒を使用する.これらは本将棋と比べて可能な局面数が少ない.しかしながら現在のところまだこれらは完全解析されていない. 完全解析されているミニ将棋として,どうぶつしょうぎやアンパンマンはじめてしょうぎ[9] がある.どうぶつしょうぎはサイズ 3×4 の盤と,ライオン,象,キリン,ひよこの 4 種類の駒を使用し,アンパンマンはじめてしょうはサイズ 3×5 の盤とアンパンマン(バイキンマン),食パンマン(ホラーマン),カレーパンマン(ドキンちゃん)の 3 種類の駒を使用する幼児向けのミニ将棋である.どうぶつしょうぎは完全解析により双方最善手を指した場合,78 手で後手が勝つことが判明している[10].また,アンパンマンはじめてしょうぎは,双方最善手を指すと千日手で引き分けとなることが判明している[11].

1.2 ゲーム AI の手法

可能な局面数が多いゲームに対して完全解析を行うことは困難である.そのようなゲームに対しては完全な最善手を得ることはできないが,局面の評価値計算,定跡データベース,一定手先の先読み,終盤での詰み読み,モンテカルロ法,機械学習などを用いてより有利だと思われる手を選択することができる.

局面の評価値計算とは,局面を判断するための指標である評価値を導出することである.コンピュータ将棋の場合のパラメータは,一般的に,成り駒を含めた駒の価値や,相手の駒同士の連携度,駒の可動範囲,攻め駒と自分の玉との距離などの相対的な位置関係などを数値化した玉の安全度などが用いられている.AI の強さは評価関数の作り方に応じて決まるため,評価関数はできるだけ戦局を適切に評価できるように工夫して作る必要がある.

定跡データベースとは,プロ棋士などが指した実践譜をもとに編成したデータベースのことである.このデータベースを使用することでより強いAIになる.しかし,相手があえて定跡以外の手を指すなどして,データベースにない局面が出てきたときにはこの手法は使えない.

一定手数の先読みとは,一定手数を先読みすることによりミニマックス法やαβ法を用いて最善の局面となる手を指させることである.たとえば,先読みの深さを3とし,局面の分岐数をxとした場合,3^x個の局面数を評価し,その中から最善の手を指すことができる.一般に先読みする手数が多いほど強いAIとなるが,先読み手数の増加に伴い探索時間が指数的に増えるため,適度に枝切りをして探索範囲を減らす工夫をする必要がある.

終盤での詰み読みとは，ゲーム終盤においてそこから詰みでの指し手を読み切ることである．この手法を用いる場合，どのタイミングで詰み読みを始めるのか，何手詰まで読むのかが重要なポイントとなる．通常の探索と詰み読みは異なった処理を行うため，それぞれどの程度の処理時間を掛けるのかを決めなければならない.

モンテカルロ法とは,乱数を用いたシミュレーションを何度も行うことにより近似解を求める計算手法である.解析的に解くことができない問題でも,十分な回数のシミュレーションを行うことにより,近似的に解を求めることができる.問題によっては他の数値計算手法より簡単に適用できるが,将棋では乱数で駒を動かして対局するため,高い精度を得ようとすると計算回数が膨大になってしまうという弱点もある.指し手をランダムに選択するために,将棋では,駒をタダで捨てる手など相手のミスを期待した手を選択してしまうという問題がある.

昨今注目されている手法にディープラーニング[12]がある.ディープラーニングはニューラルネットワークを利用した機械学習の手法であり,これをゲームに応用することで従来のAIの性能を超えるAIを作れる可能性がある.例えば囲碁では,α碁と呼ばれるAIが,将棋ではポナンザと呼ばれるAIがプロ棋士に勝つなど目覚ましい成績を上げている[13].

以上の手法を用いることにより,完全解析を行わなくてもある程度の強さのプログラムを作ることが可能であり,ゲームによってはプロに勝つこともできる.

1.3 京都将棋

京都将棋は,将棋に類するゲームの1つである.対戦人数は二人で行い,5x5の盤面で王以外の駒の裏表に異なる駒が書かれており,駒を動かすとその駒を裏返すという特別なルールにより一手ごとに駒の性能が変わるボードゲームである.

1.4 京都将棋に関する既知の結果

京都将棋は知名度が低く,現状ほとんど研究されていない.そのため,京都将棋の定跡等は確立しておらず,京都将棋特有の手法も未知数である.

既存の京都将棋のAIとしては,[14]がある.iPhoneのアプリケーションでコンピュータのレベルは4段階で設定が可能である.[15]はウェブアプリケーションでありネット上でプレイすることができる.

1.5 本報告書の目的

将棋に類するボードゲームは近年いくつも考案されており,またそれらに対するアプリケーションも多く作られている.しかしその内の1つである京都将棋は,知名度が低いため競技人口が少ない.そのため,研究はあまりされておらず,AIも殆ど見受けられない.

そこで本研究では,京都将棋のプログラムを作成し,それが強い AI となるための条件を検証する.

1.6 本報告書の構成

本報告書の構成は以下の通りである.まず 2 章で本研究の対象である京都将棋について説明する.続く 3 章では駒の評価値を用いたコンピュータ将棋の着手選択法について述べる.4 章では本研究で作成した京都将棋のプログラムについて述べる.5 章にて,検証の詳細・結果を示す.最後に 6 章で結論及び今後の課題を述べる.

2 京都将棋

本章では, 本研究の対象である京都将棋について説明する.

2.1 京都将棋の概要

京都将棋は,1976 年に田宮克哉が発表した,ごくあたらいし将棋である.京都銀閣将棋,京都銀閣金鶏秘譜将棋ともいう.

京都将棋は,ほぼ将棋と同様のルールだが,異なった点がいくつかある.5x5 の盤面で自陣,敵陣の区別はない.駒は「香ととと金」,「銀と角」,「金と桂」,「飛と歩」と王以外の駒の裏表に異なる駒が書かれており,駒を動かすとその駒を裏返し,一手ごとに駒の性能が変わる(図 1).図 2 にゲームの初期盤面を示す.

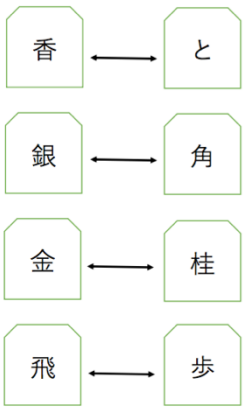

図 1 京都将棋の駒

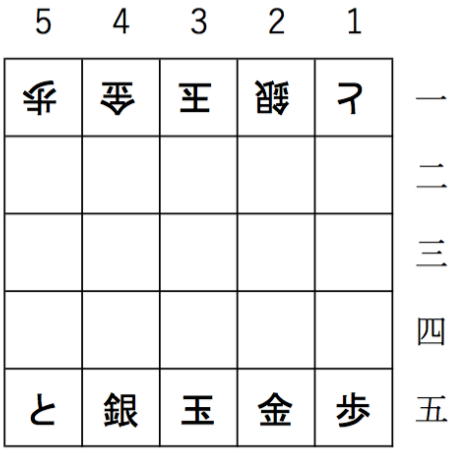

図 2 京都将棋の初期盤面

2.2 京都将棋のルール

京都将棋のルールは以下の通りである[17].

勝利条件: 通常の将棋と同様にお互いに自らの駒で相手の玉将を捕獲することを目指し,一方の玉将が相手の駒に捕獲されてしまうことが不可避な状態(詰み)となれば勝敗が決まる.また,二歩,行き所のない駒,打ち歩詰めはいずれも禁止されていない. 千日手は同一譜面 4 回で引き分けである.

駒の動き：玉以外の駒は 1 手動かすごとに元の位置・動いた先に関係なくその駒を必ず裏返す.取った駒を打つときは,裏表どちらで打ってもよい.

3　駒の評価値を用いたコンピュータ将棋の着手選択法

3.1 局面の評価値の計算

コンピュータ将棋では,各手を指した後の局面の評価値を求め,最も評価値の高い局面になるような手が選択される.局面の評価値は,局面の有利不利を決める要素,本将棋ならば駒得か駒損か,王の守りの堅さはどうか,駒同士が連携しているか,各駒の移動可能範囲はどこか,着手可能な手はいくつあるか,などのいろいろな要素をそれぞれ評価して,計算される.そして,より局面を正確にする関数ほど,性能が良い関数になると考えられる.現時点の評価値が高くても,数手先で不利になる場合もあるため,将棋のようなゲームでは,深く先を読むことが重要となる.読む深さが深くなると,探索空間は指数的に増え,計算時間が長くなってしまう.通常,将棋には持ち時間があり,時間内に次の手を指さねばならない.このため,各局面の評価値に計算に掛かる時間と,深く読むことにより読む局面数が増えることにより掛かる時間とのバランスも考慮せねばならない.あまりに複雑で,計算に時間がかかる評価関数を作ってしまうと,先読みが浅くなって,単純な評価関数で深く読むプログラムよりもかえって弱くなってしまうことすら考えられる.

将棋では,局面の評価値を求める方法の一つとして,各駒に価値を割り当て,敵味方の盤上の駒および持ち駒の価値の合計を用いる方法がよく使われる.本将棋では,プロ棋士により適切な駒の評価値がほぼ定まっているため,それを用いることができる.しかし,京都将棋では本将棋よりも盤面が小さく,また,駒の裏表が異なるために本将棋の評価値をそのまま用いることはできない.このため,京都将棋に適切な駒の評価値を求める必要がある.

3.2 ミニマックス法

数手先の局面を読んで最適となる手を選択するためには,ミニマックス法やその改良であるαβ法が用いられる.

ミニマックス法とは,探索木の葉から根に向かって評価値を求めていく手法である.ミニマックス法を用いて着手を選択する場合,まず各候補手に対して探索木を構成する.各探索木の頂点が各局面を表し,各頂点の子は 1 手先の局面を示す.葉ではその時点で盤面の評価値を求める各頂点では先手なら最大の評価値を持つ子の値,後手なら最小の評価値を持つ子の値をその頂点の評価値とする.この操作を全ての葉から根に向かって行い,根の評価値を候補手の評価値とする.ミニマックス法は,最終的に最も評価値の高い手を選択することができる手法である.しかし,この方法には欠点があり,それが探索時間によるものであるミニマックス法では,一手先を読むごとに,その手番のプレイヤーの可能な手を全て読む必要がある.

将棋では,一手で平均可能な着手の数が約 80 と言われているので,3 手先を読むには,平均的には「80*80*80=512000 手」を読む必要がある.実際には,序盤では着手の数は 30 程度と少なく,終盤では 200 程度と増えるので,終盤では「200*200*200=8000000 手」を読む必要があり,終盤になると遅くなってしまう[16].

図 3 にミニマックス法の例を示す.下から最大値,D62,E83,F90,G65 となり,次は最小値,B62,C65 となる.最後は,最大値で A は 65 となる.

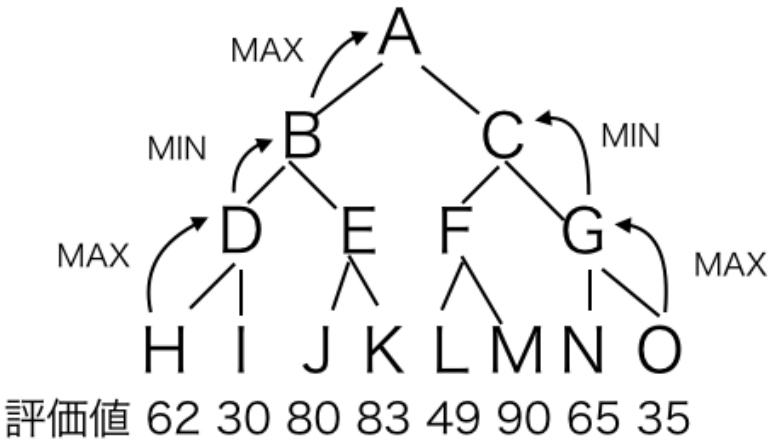

図 3 ミニマックス法の例

3.3 αβ法

αβ法とは探索アルゴリズムの 1 つでミニマックス法と同じ結果が得られるにも関わらず,理論上の計算量は,ミニマックス法比較して同じ時間でほぼ 2 倍の深さまで読むことが可能なアルゴリズムである.αβ法では,探索の際に,αカットとβカットという手法を用いる.これは,探索する必要がない枝を探索しないための工夫である.「α」「β」の範囲は,普通は「−∞」「+∞」から始めるが,この幅を縮めることで高速に探索が行われる.ただし,その場合には,必ずしも最善手順及び最善手順での評価値が得られるとは限らなくなるが,返してくる値は,α以下であれば得られる評価値の上限を示す値,β以上であれば,得られる評価値の加減を示す値になる.αβ法で,理論上の最高速度を得るには,探索の順番が完全に良い評価を返す順にソートされている必要がある[16].

図 4 にαβ法における,αカット,βカットの例を示す.D が 4 だとわかった時点で,C の値は 4 以下となり,B の値を超えないことが分かるためそれ以降を探索しない.また J が 8 だとわかった時点で,I の値は 8 以上になり,H の値を下回らないことが分かるため,それ以降を探索しない.しかしこのαβ法で最高速度を得るには,探索の順番が完全に良い評価を返す順にソートされている必要がある.

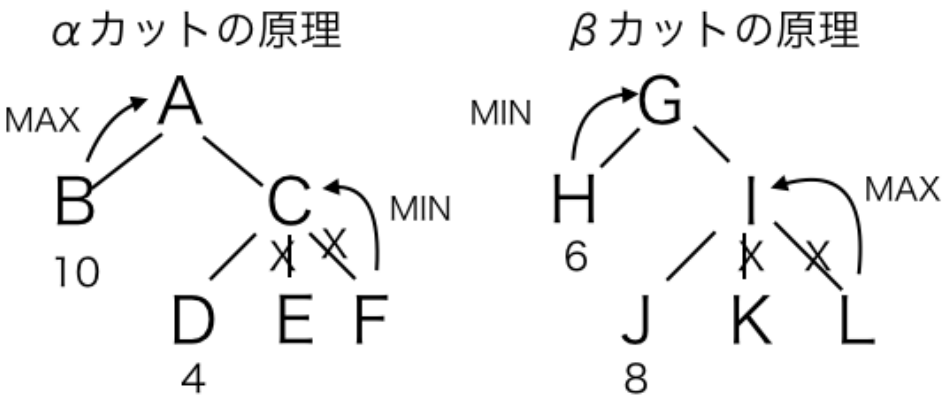

図 4 αカット,βカットの例

4　京都将棋プログラム

本章では本研究で作成した京都将棋のアプリケーションについて説明する.

本研究では,[16]の将棋プログラムをベースに,Java を用いて京都将棋プログラムを作成した.付録に本研究で作成したプログラムのソースコードを示す.

本研究で作成した京都将棋プログラムは，各駒に価値を割り当て，その価値の合計から局面の評価値を求める．そしてαβ法により 4 手先の局面まで先読みすることで着手選択している.

本研究で作成した京都将棋プログラムは,Constants クラス， KomaMoves インターフェイス,Player インターフェイス,GenerateMoves クラス,Human クラス,Koma,クラス,Kyokumen クラス,Main クラス,Position クラス,Sikou クラス,Te クラスの 11 つから成る.

4.1 Constants インターフェイス

Constants インターフェイスでは,先手の定義,後手の定義,筋を表す文字列の定義,段を表す文字列の定義の各種定数の定義を行っている.

4.2 Player インターフェイス

Player インターフェイスでは,与えられた局面から次の一手を返す関数 getNextTe()を定義する.

4.3 KomaMoves インターフェイス

KomaMoves インターフェイスは,駒の動くことのできる方向を表す.盤面上の動きで動ける方向を 8 方向と定義し,canMove という変数にてそれぞれの駒がその方向へ動けるかどうかを true なら動ける,false なら移動できないと表す.同様に canJump という変数で飛車や角,桂馬がその方向に飛べるかを true,false で表す.

4.4 GenerateMoves クラス

GenerateMoves クラスでは,合法手の生成を行う.表 1 に各メソッドについてまとめる.

表 1 GenerateMoves クラスのメソッド

メソッド	処理内容
removeSelfMate(Kyokumen k,Vecter v)	各手について,自分の玉に大手がかかってないかチェックし,王手がかかっている手は取り除く.
addTe(Kyokumen k,Vecter v,int teban,int koma,int from,int to)	与えられた Vecter に,手番,駒の種類,移動元,移動先と成るを生成した手を追加する.
generateLegalMoves(Kyokumen k)	与えられた局面における合法手を生成する.

4.5 Human クラス

Human クラスには,まず,移動元の駒の位置を 2 桁の整数で入力し,次に移動先の駒の位置を 2 桁の整数で入力する.駒を打つときには,例えば,3 三歩打であれば 0133 と入力する.投了する際には,%TORYO と入力する.合法手の一覧を出力する際には p を入力する。上記のルールに沿って,手を読み込む関数 getNextTe を実装する.

4.6 Koma クラス

Koma クラスは,駒の種類を定義する．表 2 に各メソッドについてまとめる．

表 2 Koma クラスのメソッド

メソッド	処理内容
isSente(int koma)	先手の駒かどうかの判断する．
isGote(int koma)	後手の駒かどうかの判断する．
isSelf(int teban,int koma)	手番から見て自分の細かどうか判断する．
isEnemy(int teban,int koma)	手番から見て相手の細かどうか判断する．
getKomashu(int koma)	駒の種類の取得をする．
toBanString(int koma)	盤面の表示をする.先手の駒には↑,後手の駒には↓を頭に追加する．
toString(int koma)	持ち駒,手などの表示をする．
canPromote(int koma)	駒がなれるかどうかを表す．

4.7 Kyokumen クラス

Kyokumen クラスは,盤面や持ち駒など局面を表現する．表 3 に各メソッドについてまとめる．

表 3 Kyokumen クラスのメソッド

メソッド	処理内容
clone()	局面のコピーを行う．
equals(Object o)	局面が同一かどうか.
equals(Kyokumen k)	局面が同一かどうか.
get(int p)	ある位置にある駒を取得する．
put(int p,int koma)	ある位置にある駒を置く．
move(Te te)	与えられた手で一手進める．
back(Te te)	与えられた手で一手戻す．
initKingPos()	kingS,kinfG を初期化する．
searchGyoku(int teban)	玉の位置を返す．
initEval()	初期化した際に,局面を評価する関数
evaluate()	局面を評価する関数.
toString()	局面を表示用に文字列化する．

4.8 Main クラス

Main クラスは,実行クラスである．

4.9 Position クラス

Position クラスは駒の位置を表す.各駒の位置は段と筋をそれぞれ int 型で表現する．表 5 に各メソッドについてまとめる．

表 5Position クラスのメソッド

メソッド	処理内容
Equals(Position p)	同一性比較用メソッド.
add(int diffSuji,int diffDan)	ある方向への動きを行う．

sub(int diffSuji,int diffDan)	ある方向への逆向きの動きを行う.
add(int direct)	ある方向への動きを行う.
Sub(int direct)	ある方向への逆向きの動きを行う.

4.10 Sikou クラス

Sikou クラスはコンピュータの思考ルーチンである.negaMax 様式のα β法を使用している.

4.11 Te クラス

Te クラスは手を表現する. 表 7 に各メソッドについてまとめる.

表 6 Te クラスのメソッド

メソッド	処理内容
Te(int _koma,int _from,int _to,Boolean _promote,int _capture)	コンストラクタ.
equals(Te te)	局面が同一かどうか.
equals(Object _te)	局面が同一かどうか.
Clone()	局面のコピーを行う.
toString()	手を文字列で表現する.

5 京都将棋における駒の評価値の検証

本研究で作成した京都将棋プログラムは,各駒に価値を割り当て,その価値の合計から局面の評価値を求めている. しかし,3.1 節で述べた通り,京都将棋では適切な駒の評価値は定まっていない. そこで本研究では京都将棋対戦 AI 作成し,駒の評価値を変化させながら,AI 同士で対戦させることにより, 京都将棋における最適な駒の評価値を求める. 通常の将棋の駒の評価値を裏表で平均を取った CPU(以下平均 CPU)と,対戦ごとに評価値を変動させる CPU(以下変動 CPU)の 2 つの異なった評価値を持つ CPU 作成し先手後手 100 回ずつ対戦し評価値を決める.目標は勝率 70%とする.

表 7 検証結果

変動 CPU の駒の評価値					変動 CPU 先手				変動 CPU 後手			
香と	銀角	金桂	飛歩	玉	勝	負	分	勝率	勝	負	分	勝率
200	1200	1200	1200	10000	0	0	100	−	54	46	0	54%
700	1200	1200	1200	10000	4	22	74	15%	3	97	0	3%
1700	1200	1200	1200	10000	21	76	3	21%	4	94	2	4%
2200	1200	1200	1200	10000	1	99	0	1%	83	16	1	83%
1200	200	1200	1200	10000	30	34	36	46%	87	8	5	91%
1200	700	1200	1200	10000	9	89	2	9%	1	2	97	33%
1200	1700	1200	1200	10000	24	30	46	44%	30	49	21	37%
1200	2200	1200	1200	10000	0	0	100	−	32	9	59	78%
1200	1200	200	1200	10000	0	0	100	0%	84	15	1	84%
1200	1200	700	1200	10000	32	39	29	45%	24	48	28	33%
1200	1200	1700	1200	10000	0	98	2	0%	28	56	16	33%
1200	1200	2200	1200	10000	37	55	8	40%	0	100	0	0%
1200	1200	1200	200	10000	0	0	100	−	75	7	18	91%
1200	1200	1200	700	10000	15	11	74	57%	33	55	12	37%
1200	1200	1200	1700	10000	31	68	1	31%	0	0	100	−
1200	1200	1200	2200	10000	0	23	77	0%	0	0	100	−
600	600	1200	1200	10000	0	0	100	−	13	87	0	13%
600	1200	600	1200	10000	0	0	100	−	100	0	0	100
600	1200	1200	600	10000	0	0	100	−	66	34	0	66%
1200	600	600	1200	10000	0	0	100	−	24	25	51	48%
1200	600	1200	600	10000	44	37	19	54%	53	38	9	58%
1200	1200	600	600	10000	0	0	100	−	5	58	37	7%
600	600	600	1200	10000	92	7	1	92%	93	7	0	93%
600	600	1200	600	10000	0	0	100	−	99	1	0	99%
600	1200	600	600	10000	0	0	100	−	99	1	0	99%
1200	600	600	600	10000	0	0	100	−	99	1	0	99%
1200	1700	1200	2200	10000	65	34	2	65%	60	33	7	64%
600	1700	1200	2200	10000	54	35	11	60%	50	29	21	63%
1200	1700	600	2200	10000	49	40	11	55%	47	44	9	51%

表 7 に検証に用いた駒の評価値, 平均的な評価値を持つ CPU との対戦結果を示す. 表 7 中の勝ち負け数は変動 CPU から見た勝ち負けである. 勝率の目標は勝率 70%に置いたが, 表 7 中に先手後手共に 70%を越えるものは無い.

勝率 p の勝負を N回行った場合, 標準偏差は以下の式で与えられる.

$$\sqrt{N \cdot p \cdot (1-p)}$$

$p = 0.5$ と仮定した場合, $N = 100$ なら標準偏差は

$$\sqrt{100 \cdot 0.5 \cdot 0.5} = 5.0$$

であるので, 試行回数 100 回の場合, 危険率 95%の信頼区間は 50±9.8%となる. 従って, 勝率 6 割以上なら有意に高いとみなせる.

表 7 より, 先手後手共に勝率 6 割を超え, かつ最も勝率が高いのは, 香と : 1200, 銀角 : 1700, 金桂 : 1200, 飛歩 2200 のときである. このときの勝率は先手後手共に目標で 70%を越えなかったものの 65%程度と有意に高いと言え, よって, 京都将棋に最適な評価値であると考えられる.

6 結論・今後の課題

本研究で作成した京都将棋プログラムを作成し, できる限り京都将棋に適切な駒の評価値を割り出した. 検証した結果, 相手の CPU に対して勝率は 65%と, 目標の 70%をこえることはできなかった. よってプログラムに改善する必要があると考えられる. 改善点として, 遺伝的アルゴリズムの採用をすることで対戦相手によってより多彩な戦略を持たせることにより勝率の安定などが挙げられる. また, 昨今注目を集めているディープラーニングを取り入れ強さの上限を上げることでさらに多くのプレイヤーに棋力を合わせられることにつながると考えられる. 他にも前半と後半で評価基準を変えることで, より戦術が広がると考えられる.

謝辞

本研究を作成するにあたり、指導教員の石水隆講師から、丁寧かつ熱心なご指導を賜りました。ここに感謝の意を表します。

参考文献

[1] 田中哲郎，計算機と数学　ゲームの解決，数学 65 巻 1 号，pp.93-102(2013)，https://www.jstage.jst.go.jp/article/sugaku/65/1/65_0651093/_pdf/-char/ja

[2] Jonos Wagner and Istvan Virag,Solving renju,ICGA Journal,Vol.24,No.1,PP.30-35(2001), http://www.sze.hu/~gtakacs/download/wagnervirag_2001.pdf

[3] Jonathan Schaeffer,Neil Burch,Yngvi Bjorsson,Akihiro Kishimoto,Martin Muller,Robert Lake,Paul Lu,and Steve Suphen,Checkers is solved,Science Vol.317,No.5844,pp.1518-1522(2007), http://science.sciencemag.org/content/sci/317/5844/1518.full.pdf

[4] Joel Feinstein,Amenor Wins World 6x6 Championships!,Forty billion noted under the tree(July 1993),pp.6-8,British Othello Federation’s newsletter.,(1993)

[5] 清慎一,川嶋俊:探索プログラムによる四路盤囲碁の解,情報処理学会研究報告,GI 2000(98), pp.69―76 (2000) http://id.nii.ac.jp/1001/00058633/

[6] Eric C.D. van der Welf,H.Jaap van den Herik,and Jos W.H.M.Uiterwijk,Solving Go on Small Boards:ICGAJournal,Vol.26,No.2,pp.92-107(2003).

[7] 日本 5 五将棋連盟, http://www.geocities.co.jp/Playtown-Spade/8662/

[8] 「ごろごろどうぶつしょうぎ」発売開始!,お知らせ,日本将棋連盟,2012 年 11 月 26 日(2012) https://www.shogi.or.jp/news/2012/11/post_652.html

[9] アンパンマンはじめてしょうぎ，セガトイズ(2012),https://www.segatoys.co.jp/anpan/product/popup/_legacy/learn/06.html

[10] 田中哲郎:「どうぶつしょうぎ」の完全解析,情報処理学会研究報告,Vol.2009-GI-22 No.3, pp.1-8 (2009), http://id.nii.ac.jp/1001/00062415/

[11] 塩田好,石水隆,山本博史:「アンパンマンはじめてしょうぎ」の完全解析,2013 年度 情報処理学会関西支部支部大会 講演論文集,(2013),https://ipsj.ixsq.nii.ac.jp/ej/?action=pages_view_main&active_action=repository_view_main_item_detail&item_id=96814&item_no=1&page_id=13&block_id=8

[12] 藤田一弥,高原歩夢:実装ディープラーニング,オーム社(2016)

[13] 伊藤毅志,村松正和:ディープラーニングを用いたコンピュータ囲碁～Alpha Go の技術と展望～,情報処理学会研究報告,Vol.57,No.4,pp.335-337,情報処理学会(2016),http://id.nii.ac.jp/1001/00158059/

[14] 京都将棋, Nekomado Co. Ltd(2012), https://itunes.apple.com/jp/app/京都将棋/id1037596970?mt=8

[15] 京都将棋 | 将棋ゲームの時間(2012), https://syouginojikan.web.fc2.com/kyouto.html

[16] 池泰弘：Java 将棋のアルゴリズム,工学社(2007)

[17] 京都将棋,株式会社幻冬舎エデュケーション(2014)

付録　ソースプログラム

以下に本研究で作成したプログラムのソースを示す.

```
package syougi6;

// 各種定数の定義
public interface Constants {
  // 「先手」の定義
  public final static int SENTE=1<<4;
  // 「後手」の定義
  public final static int GOTE =1<<5;

  // 筋を表す文字列の定義
  public final static String sujiStr[]={
    "","１","２","３","４","５"
  };

  // 段を表す文字列の定義
  public final static String danStr[]={
    "","一","二","三","四","五"
  };


}

package syougi6;
import java.util.Vector;

public class GenerateMoves implements Constants,KomaMoves {

  // 各手について、自分の玉に王手がかかっていないかどうかチェックし、
  // 王手がかかっている手は取り除く。
  public static Vector removeSelfMate(Kyokumen k,Vector v) {
    Vector removed=new Vector();
    for(int i=0;i<v.size();i++) {
      // 手を取り出す。
      Te te=(Te)v.elementAt(i);

      // その手で１手進めてみる
```

```
Kyokumen test=(Kyokumen)k.clone();
test.move(te);

// 自玉を探す

int gyokuPosition=test.searchGyoku(k.teban);

// 王手放置しているかどうかフラグ
boolean isOuteHouchi=false;

// 玉の周辺（１２方向）から相手の駒が利いていたら、その手は取り除く
for(int direct=0;direct<12 && !isOuteHouchi;direct++) {
  // 方向の反対方向にある駒を取得
  int pos=gyokuPosition;
  pos-=diff[direct];
  int koma=test.get(pos);
  // その駒が敵の駒で、玉方向に動けるか？
  if (Koma.isEnemy(test.teban,koma) && canMove[direct][koma]) {
    // 動けるなら、この手は王手を放置しているので、
    // この手は、removedに追加しない。
    isOuteHouchi=true;
    break;
  }
}

// 玉の周り（８方向）から相手の駒の飛び利きがあるなら、その手は取り除く
for(int direct=0;direct<8 && !isOuteHouchi;direct++) {
  // 方向の反対方向にある駒を取得
  int pos=gyokuPosition;
  int koma;
  // その方向にマスが空いている限り、駒を探す
  for(pos-=diff[direct],koma=test.get(pos);
   koma!=Koma.WALL;pos-=diff[direct],koma=test.get(pos)) {
    // 味方駒で利きが遮られているなら、チェック終わり。
    if (Koma.isSelf(test.teban,koma)) break;
    // 遮られていない相手の駒の利きがあるなら、王手がかかっている。
    if (Koma.isEnemy(test.teban,koma) && canJump[direct][koma]) {
      isOuteHouchi=true;
      break;
```

```
        }
        // 敵駒で利きが遮られているから、チェック終わり。
        if (Koma.isEnemy(test.teban,koma)) {
          break;
        }
      }
    }
    if (!isOuteHouchi) {
      removed.add(te);
    }
  }
  return removed;
}

// 与えられたVectorに、手番、駒の種類、移動元、移動先を考慮して、
// 成る・不成りを判断しながら生成した手を追加する。
public static void addTe(Kyokumen k,Vector v,int teban,int koma,int from,int to) {
  if (teban==SENTE) {
      //先手番
      Te te=new Te(koma,from,to,true,k.get(to));
      v.add(te);
  } else {
    // 後手番
      Te te=new Te(koma,from,to,true,k.get(to));
      v.add(te);
  }
}


// 与えられた局面における合法手を生成する。
public static Vector generateLegalMoves(Kyokumen k) {
  Vector v=new Vector();

  // 盤上の手番の側の駒を動かす手を生成
  for(int suji=0x10;suji<=0x50;suji+=0x10) {
    for(int dan=1;dan<=5;dan++) {
      int from=dan+suji;
      int koma=k.get(from);
      // 自分の駒であるかどうか確認
```

```
    if (Koma.isSelf(k.teban,koma)) {
      // 各方向に移動する手を生成
      for(int direct=0;direct<12;direct++) {
        if (canMove[direct][koma]) {
          // 移動先を生成
          int to=from+diff[direct];
          // 移動先は盤内か？
          if (1<=(to>>4) && (to>>4)<=5 && 1<=(to&0x0f) && (to&0x0f)<=5) {
            // 移動先に自分の駒がないか？
            if (Koma.isSelf(k.teban,k.get(to))) {
              // 自分の駒だったら、次の方向を検討
              continue;
            }
            // 成る・不成りを考慮しながら、手をvに追加
            addTe(k,v,k.teban,koma,from,to);
          }
        }
      }
      // 各方向に「飛ぶ」手を生成
      for(int direct=0;direct<8;direct++) {
        if (canJump[direct][koma]) {
          // そちら方向に飛ぶことが出来る
          for(int i=1;i<9;i++) {
            // 移動先を生成
            int to=from+diff[direct]*i;
            // 行き先が盤外だったら、そこには行けない
            if (k.get(to)==Koma.WALL) break;
            // 行き先に自分の駒があったら、そこには行けない
            if (Koma.isSelf(k.teban,k.get(to))) break;
            // 成る・不成りを考慮しながら、手をvに追加
            addTe(k,v,k.teban,koma,from,to);
            // 空き升でなければ、ここで終わり
            if (k.get(to)!=Koma.EMPTY) break;
          }
        }
      }
    }
  }
}
```

```
    // 手番の側の駒を打つ手を生成

    // まず、駒の種類でループ
    for(int i=Koma.FU;i<=Koma.TO;i++) {
      // 打つ駒は、手番の側の駒
      int koma=i|k.teban;
      // その駒を持っているか？
      if (k.hand[koma]>0) {
        // 持っている。
        int komashu=Koma.getKomashu(koma);
        // 盤面の各升目でループ
        for(int suji=0x10;suji<=0x50;suji+=0x10) {
          for(int dan=1;dan<=5;dan++) {
            // 移動元…駒を打つ手は、0
            int from=0;
            // 移動先、駒を打つ場所
            int to=suji+dan;
            // 空き升でなければ、打つ事は出来ない。
            if (k.get(to)!=Koma.EMPTY) {
              continue;
            }

            // 手の生成…駒を打つ際には、常に不成で、取る駒もなしである。
          Te te=new Te(koma,from,to,false,Koma.EMPTY);

//            if(komashu==Koma.KY){
//              te=new Te(Koma.TO,from,to,false,Koma.EMPTY);
//               v.add(te);
//              te=new Te(Koma.KY,from,to,false,Koma.EMPTY);
//               v.add(te);
//            }else if(komashu==Koma.GI){
//              te=new Te(Koma.KA,from,to,false,Koma.EMPTY);
//               v.add(te);
//              te=new Te(Koma.GI,from,to,false,Koma.EMPTY);
//               v.add(te);
//            }else if(komashu==Koma.KI){
//              te=new Te(Koma.KE,from,to,false,Koma.EMPTY);
```

```
//              v.add(te);
//             te=new Te(Koma.KI,from,to,false,Koma.EMPTY);
//              v.add(te);
//           }else if(komashu==Koma.HI){
//             te=new Te(Koma.FU,from,to,false,Koma.EMPTY);
//              v.add(te);
//             te=new Te(Koma.HI,from,to,false,Koma.EMPTY);
//              v.add(te);
//           }else{
//           }
            // 駒を打つ手が可能なことが分かったので、合法手に加える。
           v.add(te);
          }
        }
      }
    }

    // 生成した各手について、指してみて
    // 自分の玉に王手がかかっていないかどうかチェックし、
    // 王手がかかっている手は取り除く。
    v=removeSelfMate(k,v);

    return v;
  }
}

package syougi6;

//import javax.swing.text.StyledEditorKit.ForegroundAction;

// 駒
class Koma implements Constants,Cloneable {
  // 駒の種類の定義
  public static final int EMPTY=0;          // 「空」
  public static final int EMP=EMPTY;        // 「空」の別名。

  public static final int PROMOTE[]=
         {0,0,0,0,0,0,0,0,// 先手でも後手でもない駒
         0,0,0,0,0,0,0,0,// 先手でも後手でもない駒
```

```
        0,6,6,2,2,-2,-2,-6,// 空、先手の歩香桂銀金角飛
        -6,0,0,0,0,0,0,0,// 先手の王、と杏圭全　馬竜
        0,6,6,2,2,-2,-2,-6,// 空、後手の歩香桂銀金角飛
        -6,0,0,0,0,0,0,0,// 後手の王、と杏圭全　馬竜
         };


  // 「成り」フラグ

public static final int FU= 1;          // 「歩」
public static final int KY= 2;          // 「香車」
public static final int KE= 3;          // 「桂馬」
public static final int GI= 4;          // 「銀」
public static final int KI= 5;          // 「金」
public static final int KA= 6;          // 「角」
public static final int HI= 7;          // 「飛車」
public static final int TO= 8;                  // 「と金」
public static final int OU= 9;          // 「玉将」


  public static final int SFU=SENTE+FU;    // 「先手の歩」＝「歩」＋「先手」
  public static final int SKY=SENTE+KY;    // 「先手の香」
  public static final int SKE=SENTE+KE;    // 「先手の桂」
  public static final int SGI=SENTE+GI;    // 「先手の銀」
  public static final int SKI=SENTE+KI;    // 「先手の金」
  public static final int SKA=SENTE+KA;    // 「先手の角」
  public static final int SHI=SENTE+HI;    // 「先手の飛」
  public static final int STO=SENTE+TO;    // 「先手のと金」
  public static final int SOU=SENTE+OU;    // 「先手の玉」

  public static final int GFU=GOTE +FU;    // 「後手の歩」＝「歩」＋「後手」
  public static final int GKY=GOTE +KY;    // 「後手の香」
  public static final int GKE=GOTE +KE;    // 「後手の桂」
  public static final int GGI=GOTE +GI;    // 「後手の銀」
  public static final int GKI=GOTE +KI;    // 「後手の金」
  public static final int GKA=GOTE +KA;    // 「後手の角」
  public static final int GHI=GOTE +HI;    // 「後手の飛」
  public static final int GTO=GOTE +TO;    // 「後手のと金」
  public static final int GOU=GOTE +OU;    // 「後手の玉」
```

```
public static final int WALL=64;          // 盤の外を表すための定数



// 先手の駒かどうかの判定
static public boolean isSente(int koma) {
  return (koma & SENTE)!=0;
}

// 後手の駒かどうかの判定
static public boolean isGote(int koma) {
  return (koma & GOTE)!=0;
}

// 手番から見て自分の駒かどうか判定
static public boolean isSelf(int teban,int koma) {
  if (teban==SENTE) {
    return isSente(koma);
  } else {
    return isGote(koma);
  }
}

// 手番から見て相手の駒かどうか判定
static public boolean isEnemy(int teban,int koma) {
  if (teban==SENTE) {
    return isGote(koma);
  } else {
    return isSente(koma);
  }
}

// 駒の種類の取得
static public int getKomashu(int koma) {
  // 先手後手のフラグをビット演算でなくせば良い。
  return koma & 0x0f;
}
```

```
// 駒の文字列化用の文字列
static public final String komaString[]={
  "  ",
  "歩",
  "香",
  "桂",
  "銀",
  "金",
  "角",
  "飛",
  "と",
  "王",
  "杏",
  "圭",
  "全",
  "",
  "馬",
  "竜"
};

// 駒の文字列化…盤面の表示用
static public String toBanString(int koma) {
  if ( koma==EMPTY ) {
    return "   ";
  } else if ( (koma & SENTE) !=0 ) {
    // 先手の駒には、"↑"を頭に追加
    return "↑"+komaString[getKomashu(koma)];
  } else {
    // 後手の駒には、"↓"を頭に追加
    return "↓"+komaString[getKomashu(koma)];
  }
}

// 駒の文字列化…持ち駒、手などの表示用
static public String toString(int koma) {
  return komaString[getKomashu(koma)];
}

// 駒が成れるかどうかを表す
```

```
  public static final boolean canPromote[]={
     false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
     false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
     false, true, true, true, true,false, true, true,// 空、先手の歩香桂銀金角飛
     false,false,false,false,false,false,false,false,// 先手の王、と杏圭全　馬竜
     false, true, true, true, true,false, true, true,// 空、後手の歩香桂銀金角飛
     false,false,false,false,false,false,false,false // 後手の王、と杏圭全　馬竜
  };

  static public boolean canPromote(int koma) {
    return canPromote[koma];
  }
}

package syougi6;

public interface KomaMoves {
  // 通常の８方向の定義(盤面上の動き)
  //
  //  5   6  7
  //     ↑
  //  3←駒→4
  //     ↓
  //  2   1  0
  //
  // 桂馬飛びの方向の定義(盤面上の動き)
  //
  //   8    9
  //
  //     桂
  //
  //   11  10

  // 方向の定義に沿った、「段」の移動の定義
  public static final int diffDan[]={
     1, 1, 1, 0, 0,-1,-1,-1,-2,-2, 2, 2
  };

  // 方向の定義に沿った、「筋」の移動の定義
```

```
public static final int diffSuji[]={
  -1, 0, 1, 1,-1, 1, 0,-1, 1,-1,-1, 1
};

// 方向の定義に沿った、「移動」の定義
public static final int diff[]={
  diffSuji[0]*16+diffDan[0],
  diffSuji[1]*16+diffDan[1],
  diffSuji[2]*16+diffDan[2],
  diffSuji[3]*16+diffDan[3],
  diffSuji[4]*16+diffDan[4],
  diffSuji[5]*16+diffDan[5],
  diffSuji[6]*16+diffDan[6],
  diffSuji[7]*16+diffDan[7],
  diffSuji[8]*16+diffDan[8],
  diffSuji[9]*16+diffDan[9],
  diffSuji[10]*16+diffDan[10],
  diffSuji[11]*16+diffDan[11]
};


// ある方向にある駒が動けるかどうかを表すテーブル。
// 添え字の１つめが方向で、２つめが駒の種類である。
// 香車や飛車、角などの一直線に動く動きについては、後述のcanJumpで表し、
// このテーブルではfalseとしておく。
public static final boolean canMove[][]={
  // 方向 0 斜め左下への動き
  {
   false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
   false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
   false,false,false,false, true,false,false,false,// 空、先手の歩香桂銀金角飛
    false,true,false,false,false,false,false, true,// 先手の王、と杏圭全　馬竜
   false,false,false,false, true, true,false,false,// 空、後手の歩香桂銀金角飛
    true, true, true, true, true, true,false, true // 後手の王、と杏圭全　馬竜
  },
  // 方向 1 真下への動き
  {
   false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
   false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
```

```
 false,false,false,false,false, true,false,false,// 空、先手の歩香桂銀金角飛
  true, true, true, true, true,false, true,false,// 先手の王、と杏圭全　馬竜
 false, true,false,false, true, true,false,false,// 空、後手の歩香桂銀金角飛
  true, true, true, true, true,false, true,false // 後手の王、と杏圭全　馬竜
},
// 方向 2 斜め右下への動き
{
 false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
 false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
 false,false,false,false, true,false,false,false,// 空、先手の歩香桂銀金角飛
  false,true,false,false,false,false,false, true,// 先手の王、と杏圭全　馬竜
 false,false,false,false, true, true,false,false,// 空、後手の歩香桂銀金角飛
  true, true, true, true, true, true,false, true // 後手の王、と杏圭全　馬竜
},
// 方向 3 左への動き
{
 false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
 false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
 false,false,false,false,false, true,false,false,// 空、先手の歩香桂銀金角飛
  true, true, true, true, true,false, true,false,// 先手の王、と杏圭全　馬竜
 false,false,false,false,false, true,false,false,// 空、後手の歩香桂銀金角飛
  true, true, true, true, true,false, true,false // 後手の王、と杏圭全　馬竜
},
// 方向 4 右への動き
{
 false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
 false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
 false,false,false,false,false, true,false,false,// 空、先手の歩香桂銀金角飛
  true, true, true, true, true,false, true,false,// 先手の王、と杏圭全　馬竜
 false,false,false,false,false, true,false,false,// 空、後手の歩香桂銀金角飛
  true, true, true, true, true,false, true,false // 後手の王、と杏圭全　馬竜
},
// 方向 5 斜め左上への動き
{
 false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
 false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
 false,false,false,false, true, true,false,false,// 空、先手の歩香桂銀金角飛
  true, true, true, true, true,false,false, true,// 先手の王、と杏圭全　馬竜
 false,false,false,false, true,false,false,false,// 空、後手の歩香桂銀金角飛
```

```
 false,true,false,false,false,false,false, true // 後手の王、と杏圭全　馬竜
},
// 方向 6 真上への動き
{
false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
false, true,false,false, true, true,false,false,// 空、先手の歩香桂銀金角飛
 true, true, true, true, true,false, true,false,// 先手の王、と杏圭全　馬竜
false,false,false,false,false, true,false,false,// 空、後手の歩香桂銀金角飛
 true, true, true, true, true,false, true,false // 後手の王、と杏圭全　馬竜
},
// 方向 7 斜め右上への動き
{
 false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
 false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
 false,false,false,false, true, true,false,false,// 空、先手の歩香桂銀金角飛
  true, true, true, true, true,false,false, true,// 先手の王、と杏圭全　馬竜
 false,false,false,false, true,false,false,false,// 空、後手の歩香桂銀金角飛
  false,true,false,false,false,false,false, true // 後手の王、と杏圭全　馬竜
},
// 方向 8 先手の桂馬飛び
{
 false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
 false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
 false,false,false, true,false,false,false,false,// 空、先手の歩香桂銀金角飛
 false,false,false,false,false,false,false,false,// 先手の王、と杏圭全　馬竜
 false,false,false,false,false,false,false,false,// 空、後手の歩香桂銀金角飛
 false,false,false,false,false,false,false,false // 後手の王、と杏圭全　馬竜
},
// 方向 9 先手の桂馬飛び
{
 false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
 false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
 false,false,false, true,false,false,false,false,// 空、先手の歩香桂銀金角飛
 false,false,false,false,false,false,false,false,// 先手の王、と杏圭全　馬竜
 false,false,false,false,false,false,false,false,// 空、後手の歩香桂銀金角飛
 false,false,false,false,false,false,false,false // 後手の王、と杏圭全　馬竜
},
// 方向10 後手の桂馬飛び
```

```
    {
     false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
     false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
     false,false,false,false,false,false,false,false,// 空、先手の歩香桂銀金角飛
     false,false,false,false,false,false,false,false,// 先手の王、と杏圭全　馬竜
     false,false,false, true,false,false,false,false,// 空、後手の歩香桂銀金角飛
     false,false,false,false,false,false,false,false // 後手の王、と杏圭全　馬竜
    },
    // 方向11 後手の桂馬飛び
    {
     false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
     false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
     false,false,false,false,false,false,false,false,// 空、先手の歩香桂銀金角飛
     false,false,false,false,false,false,false,false,// 先手の王、と杏圭全　馬竜
     false,false,false, true,false,false,false,false,// 空、後手の歩香桂銀金角飛
     false,false,false,false,false,false,false,false // 後手の王、と杏圭全　馬竜
    }
};

// ある方向にある駒が飛べるかどうかを表すテーブル。
// 添え字の１つめが方向で、２つめが駒の種類である。
// 香車や飛車、角、竜、馬の一直線に動く動きについては、こちらで表す。
static final public boolean canJump[][]={
  // 方向 0 斜め左下への動き
  {
   false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
   false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
   false,false,false,false,false,false, true,false,// 空、先手の歩香桂銀金角飛
   false,false,false,false,false,false, true,false,// 先手の王、と杏圭全　馬竜
   false,false,false,false,false,false, true,false,// 空、後手の歩香桂銀金角飛
   false,false,false,false,false,false, true,false // 後手の王、と杏圭全　馬竜
  },
  // 方向 1 真下への動き
  {
   false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
   false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
   false,false,false,false,false,false,false, true,// 空、先手の歩香桂銀金角飛
   false,false,false,false,false,false,false, true,// 先手の王、と杏圭全　馬竜
   false,false, true,false,false,false,false, true,// 空、後手の歩香桂銀金角飛
```

```
 false,false,false,false,false,false,false, true // 後手の王、と杏圭全　馬竜
},
// 方向 2 斜め右下への動き
{
 false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
 false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
 false,false,false,false,false,false, true,false,// 空、先手の歩香桂銀金角飛
 false,false,false,false,false,false, true,false,// 先手の王、と杏圭全　馬竜
 false,false,false,false,false,false, true,false,// 空、後手の歩香桂銀金角飛
 false,false,false,false,false,false, true,false // 後手の王、と杏圭全　馬竜
},
// 方向 3 左への動き
{
 false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
 false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
 false,false,false,false,false,false,false, true,// 空、先手の歩香桂銀金角飛
 false,false,false,false,false,false,false, true,// 先手の王、と杏圭全　馬竜
 false,false,false,false,false,false,false, true,// 空、後手の歩香桂銀金角飛
 false,false,false,false,false,false,false, true // 後手の王、と杏圭全　馬竜
},
// 方向 4 右への動き
{
 false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
 false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
 false,false,false,false,false,false,false, true,// 空、先手の歩香桂銀金角飛
 false,false,false,false,false,false,false, true,// 先手の王、と杏圭全　馬竜
 false,false,false,false,false,false,false, true,// 空、後手の歩香桂銀金角飛
 false,false,false,false,false,false,false, true // 後手の王、と杏圭全　馬竜
},
// 方向 5 斜め左上への動き
{
 false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
 false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
 false,false,false,false,false,false, true,false,// 空、先手の歩香桂銀金角飛
 false,false,false,false,false,false, true,false,// 先手の王、と杏圭全　馬竜
 false,false,false,false,false,false, true,false,// 空、後手の歩香桂銀金角飛
 false,false,false,false,false,false, true,false // 後手の王、と杏圭全　馬竜
},
// 方向 6 真上への動き
```

```
    {
     false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
     false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
     false,false, true,false,false,false,false, true,// 空、先手の歩香桂銀金角飛
     false,false,false,false,false,false,false, true,// 先手の王、と杏圭全　馬竜
     false,false,false,false,false,false,false, true,// 空、後手の歩香桂銀金角飛
     false,false,false,false,false,false,false, true // 後手の王、と杏圭全　馬竜
    },
    // 方向 7 斜め右上への動き
    {
     false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
     false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
     false,false,false,false,false,false, true,false,// 空、先手の歩香桂銀金角飛
     false,false,false,false,false,false, true,false,// 先手の王、と杏圭全　馬竜
     false,false,false,false,false,false, true,false,// 空、後手の歩香桂銀金角飛
     false,false,false,false,false,false, true,false // 後手の王、と杏圭全　馬竜
    }
    // 桂馬の方向に飛ぶ駒はないので、以下は省略。
  };
}


package syougi6;

public interface Player {
  Te getNextTe(Kyokumen k);
}

package syougi6;

// 駒の位置を表すクラス
class Position implements Cloneable,KomaMoves {
  // 筋
  public int suji;
  // 段
  public int dan;

  // コンストラクタ
  public Position() {
```

```
  suji=0;
  dan=0;
}

// コンストラクタ
public Position(int _suji,int _dan) {
  suji=_suji;
  dan=_dan;
}

// 同一性比較用メソッド
public boolean equals(Position p) {
  return (p.suji==suji && p.dan==dan);
}
public boolean equals(Object o) {
  Position p=(Position)o;
  if (p==null) return false;
  return equals(p);
}

// コピーを返す
public Object clone() {
  return new Position(suji,dan);
}

// ある方向への動きを行う
public void add(int diffSuji,int diffDan) {
  suji+=diffSuji;
  dan+=diffDan;
}

// ある方向への逆向きの動きを行う
public void sub(int diffSuji,int diffDan) {
  suji-=diffSuji;
  dan-=diffDan;
}

// ある方向への動きを行う
public void add(int direct) {
```

```
    add(diffSuji[direct],diffDan[direct]);
  }

  // ある方向への逆向きの動きを行う
  public void sub(int direct) {
    sub(diffSuji[direct],diffDan[direct]);
  }
}

package syougi6;

public class Te implements Cloneable,Constants {
  int koma;               // どの駒が動いたか
  int from;               // 動く前の位置（持ち駒の場合、0）
  int to;                 // 動いた先の位置
  boolean promote;         // 成る場合、true 成らない場合 false
  int capture;            // 取った駒（Kyokumenのback関数で利用する）

  public Te(int _koma,int _from,int _to,boolean _promote,int _capture) {
    koma=_koma;
    from=_from;
    to=  _to;
    promote=_promote;
    capture=_capture;
  }

  public boolean equals(Te te) {
         return (te.koma==koma && te.from==from && te.to==to);
  //  return (te.koma==koma && te.from==from && te.to==to && te.promote==promote);
  }

  public boolean equals(Object _te) {
    Te te=(Te)_te;
    if (te==null) return false;
    return equals(te);
  }

  public Object clone() {
    return new Te(koma,from,to,promote,capture);
```

```
  }

  // 手を文字列で表現する。
  public String toString() {
    return sujiStr[to>>4]+danStr[to&0x0f]+
           Koma.toString(koma)+(promote?"":"")+
           (from==0?"打    ":"("+sujiStr[from>>4]+danStr[from&0x0f]+")")+
           (promote?"":"  ");

  }
}

package syougi6;
import java.util.ArrayList;
import java.util.Vector;
import java.util.Random;
import java.security.SecureRandom;

class Kyokumen implements Constants,Cloneable {

  // 盤面
  int ban[];

  // 持ち駒
  int hand[];

  // 手番
  int teban=SENTE;

  Random ran  = new Random();

  // 現在の先手からみた評価値
  int eval=0;

  // 先手玉の位置…盤外の、利きの届かないところ、(-2,-2)=-2*16-2=-34に設定。
  int kingS=-34;

  // 後手玉の位置…盤外の、利きの届かないところ、(-2,-2)=-2*16-2=-34に設定。
```

```
int kingG=-34;

public Kyokumen() {
  ban=new int[16*11];
  hand=new int[Koma.GTO+1];
  // 盤面全体を「カベ」で一旦埋める
  for(int i=0;i<16*11;i++) {
    ban[i]=Koma.WALL;
  }
  // 盤面にあたる場所を空白に設定する
  for(int suji=1;suji<=5;suji++) {
    for(int dan=1;dan<=5;dan++) {
      ban[(suji<<4)+dan]=Koma.EMPTY;
    }
  }
}

// 局面のコピーを行う
public Object clone() {
  Kyokumen k=new Kyokumen();

  // 盤面のコピー
  for(int i=0;i<16*11;i++) {
    k.ban[i]=ban[i];
  }

  // 持ち駒のコピー
  for(int i=Koma.SFU;i<=Koma.GTO;i++) {
    k.hand[i]=hand[i];
  }


  // 手番のコピー
  k.teban=teban;

  // 評価値のコピー
  k.eval=eval;
```

```
  // 玉の位置のコピー
  k.kingS=kingS;
  k.kingG=kingG;

  return k;
}

// 局面が同一かどうか
public boolean equals(Object o) {
  Kyokumen k=(Kyokumen)o;
  if (k==null) return false;
  return equals(k);
}

// 局面が同一かどうか
public boolean equals(Kyokumen k) {
  // 手番の比較
  if (teban!=k.teban) {
    return false;
  }

  // 盤面の比較
  // 各マスについて…
  for(int suji=0x10;suji<=0x50;suji+=0x10) {
    for(int dan=1;dan<=5;dan++) {
      // 盤面上の筋と段にある駒が、比較対象の盤面上の同じ位置にある駒と
      // 同じかどうか比較する。
      if (ban[suji+dan]!=k.ban[suji+dan]) {
        // 違っていたら、falseを返す。
        return false;
      }
    }
  }

  // 持ち駒の比較
  // 持ち駒の枚数を比較する。
  for(int i=Koma.SFU;i<=Koma.GTO;i++) {
    if (hand[i]!=k.hand[i]) {
```

```
      // 違っていたら、falseを返す。
      return false;
    }
  }

  // 完全に一致した。
  return true;
}

// ある位置にある駒を取得する
public int get(int p) {
  // 盤外なら、「盤外＝壁」を返す
  if (p<0 || 16*11<p) {
    return Koma.WALL;
  }
  return ban[p];
}

// ある位置にある駒を置く。
public void put(int p,int koma) {
  ban[p]=koma;
}

// 与えられた手で一手進めてみる。
public void move(Te te) {
  // 駒の行き先に駒があったなら…
  if (get(te.to)!=Koma.EMPTY) {
    // 盤面からその駒がなくなった分、評価値を減じる。
      if(teban==SENTE){
            eval-=komaValue1[get(te.to)];
            }else{
                  eval-=komaValue2[get(te.to)];
            }

    // 持ち駒にする
    if (Koma.isSente(get(te.to))) {
      // 取った駒が先手の駒なら後手の持ち駒に。
      int koma=get(te.to);
```

```
        // 成りなどのフラグ、先手・後手の駒のフラグをクリア。
        if(koma==Koma.SKY || koma==Koma.STO){
                koma=Koma.GKY;
        }else if (koma==Koma.SGI || koma==Koma.SKA) {
                        koma=Koma.GGI;
                }else if (koma==Koma.SKI || koma==Koma.SKE) {
                        koma=Koma.GKI;
                }else if(koma==Koma.SHI || koma==Koma.SFU) {
                        koma=Koma.GHI;
                }else{

                }
//        koma=koma & 0x07;
//        // 後手の駒としてのフラグをセット
//        koma=koma | GOTE;
        // 持ち駒に追加。
        hand[koma]++;
        eval+=komaValue2[koma];
      } else {
        // 取った駒が後手の駒なら先手の持ち駒に。
        int koma=get(te.to);
        // 成りなどのフラグ、先手・後手の駒のフラグをクリア
        if(koma==Koma.GKY || koma==Koma.GTO){
                koma=Koma.SKY;
        }else if (koma==Koma.GGI || koma==Koma.GKA) {
                        koma=Koma.SGI;
                }else if (koma==Koma.GKI || koma==Koma.GKE) {
                        koma=Koma.SKI;
                }else if(koma==Koma.GHI || koma==Koma.GFU) {
                        koma=Koma.SHI;
                }else{

                }

//        koma=koma & 0x07;
//        // 先手の駒としてのフラグをセット
//        koma=koma | SENTE;
        // 持ち駒に追加。
```

```
        hand[koma]++;
        eval+=komaValue1[koma];
      }
    }
    if (te.from==0) {
      // 持ち駒を打った
      // 持ち駒を一枚減らす。
      hand[te.koma]--;
    } else {
      // 盤上の駒を進めた→元の位置は、EMPTYに。
      put(te.from,Koma.EMPTY);
    }
    // 駒を移動先に進める。
    int koma=te.koma;
    if (te.promote) {
      // 「成り」の処理
      // 成る前の駒の価値を減じる

//      if(teban==SENTE){
//     eval-=komaValue1[koma];
//     }else{
//           eval-=komaValue2[koma];
//           }



      koma=koma+Koma.PROMOTE[koma]; //ここ

      // 成った後の駒の価値を加える
//      if(teban==SENTE){
//       eval+=komaValue1[koma];
//           }else{
//                 eval+=komaValue2[koma];
//           }

    }
    put(te.to,koma);
    if (te.koma==Koma.SOU) {
```

```
      kingS=te.to;
    } else if (te.koma==Koma.GOU) {
      kingG=te.to;
    }
  }

  // 与えられた手で一手戻す。
  public void back(Te te) {
    // 取った駒を盤に戻す
    put(te.to,te.capture);
    // 評価点も戻す
        if(teban==SENTE){
              eval+=komaValue1[te.capture];
               }else{
                    eval+=komaValue2[te.capture];
          }

    // 取った駒がある時には…
    if (te.capture!=Koma.EMPTY) {
      // 持ち駒に入っているはずなので、減らす。
      if (Koma.isSente(te.capture)) {
        // 取った駒が先手の駒なら後手の持ち駒に。
        int koma=te.capture;
        // 成りなどのフラグ、先手・後手の駒のフラグをクリア。
        if(koma==Koma.SKY || koma==Koma.STO){
              koma=Koma.GKY;
        }else if (koma==Koma.SGI || koma==Koma.SKA) {
                    koma=Koma.GGI;
              }else if (koma==Koma.SKI || koma==Koma.SKE) {
                    koma=Koma.GKI;
              }else if(koma==Koma.SHI || koma==Koma.SFU) {
                    koma=Koma.GHI;
              }else{

              }
//        koma=koma & 0x07;
//        // 後手の駒としてのフラグをセット
//        koma=koma | GOTE;
```

```
        // 持ち駒に追加。
        hand[koma]--;
        eval-=komaValue2[koma];
      } else {
        // 取った駒が後手の駒なら先手の持ち駒に。
        int koma=te.capture;
        // 成りなどのフラグ、先手・後手の駒のフラグをクリア。
        if(koma==Koma.GKY || koma==Koma.GTO){
              koma=Koma.SKY;
        }else if (koma==Koma.GGI || koma==Koma.GKA) {
                    koma=Koma.SGI;
              }else if (koma==Koma.GKI || koma==Koma.GKE) {
                    koma=Koma.SKI;
              }else if(koma==Koma.GHI || koma==Koma.GFU) {
                    koma=Koma.SHI;
              }else{

              }


//        koma=koma & 0x07;
//        // 先手の駒としてのフラグをセット
//        koma=koma | SENTE;
        // 持ち駒に追加。
        hand[koma]--;
        eval-=komaValue1[koma];
      }
    }

    if (te.from==0) {
      // 駒打ちだったので、持ち駒に戻す
      hand[te.koma]++;
    } else {
      // 動かした駒を元の位置に戻す。
      put(te.from,te.koma);
      if (te.promote) {
        // 成っていたので、その分の点数を計算しなおす。
        // 成った後の駒の価値を減じる
```

```
      int koma=te.koma+Koma.PROMOTE[te.koma];
       if(teban==SENTE){
            eval-=komaValue1[koma];
       }else{
             eval-=komaValue2[koma];
}

      // 成る前の駒の価値を加える
      if(teban==SENTE){
            eval+=komaValue1[te.koma];
       }else{
             eval+=komaValue2[te.koma];
 }

    }
  }
  if (te.koma==Koma.SOU) {
    kingS=te.from;
  } else if (te.koma==Koma.GOU) {
    kingG=te.from;
  }
}

// kingS,kingGを初期化する
void initKingPos() {
  // 先手と後手の玉の位置…盤外で、利きの届かない位置(-2,-2)にあたる
  // 位置で初期化しておく。
  kingS=-34;
  kingG=-34;
  // 筋、段でループ
  for(int suji=0x10;suji<=0x50;suji+=0x10) {
    for(int dan=1;dan<=5;dan++) {
      if (ban[suji+dan]==Koma.SOU) {
        // 見つかった
        kingS=suji+dan;
      }
      if (ban[suji+dan]==Koma.GOU) {
        // 見つかった
```

```
          kingG=suji+dan;
        }
      }
    }
  }

  // 玉の位置を返す
  public int searchGyoku(int teban) {
    if (teban==SENTE) {
      return kingS;
    } else {
      return kingG;
    }
  }



   int a =ran.nextInt(50);
   int b =ran.nextInt(50);
   int c =ran.nextInt(50);
   int d =ran.nextInt(50);


  // 局面を評価するための、駒の価値。
  // 先手の駒はプラス点、後手の駒はマイナス点にする。
   final int komaValue1[]={
       0,   0,   0,   0,   0,   0,   0,   0, // 何もない場所及び
       0,   0,   0,   0,   0,   0,   0,   0, // 先手でも後手でもない駒
       0, 2200, 1200, 1200, 1700, 1200, 1700, 2200, // 何もない場所、先手の歩～飛
車
    1200, 10000,   0,   0,   0,   0,   0,   0, // 先手玉、及びと～竜
       0, -2200, -1200, -1200, -1700,-1200,-1700, -2200, // 何もない場所、後手の
歩～飛車
    -1200,-10000,   0,   0,   0,   0,   0,   0 // 後手玉、及びと～竜
  };


   final int komaValue2[]={
```

```
      0,    0,    0,    0,    0,    0,    0,    0, // 何もない場所及び
      0,    0,    0,    0,    0,    0,    0,    0, // 先手でも後手でもない駒
      0, 1550+d,  900+a, 950+c, 1400+b, 950+c, 1400+b, 1550+d, // 何もない場所、先手の歩～飛車
    900+a, 10000,    0,    0,    0,    0,    0,    0, // 先手玉、及びと～竜
      0, -1550+d, -900-a, -950-c,-1400-b,-950-c,-1400-b, -1550+d, // 何もない場所、後手の歩～飛車
   -900-a, -10000,    0,    0,    0,    0,    0,    0 // 後手玉、及びと～竜
};

  // 初期化した際に、局面を評価する関数
  void initEval() {
    eval=0;
    // まず、盤面の駒から。
    for(int suji=0x10;suji<=0x50;suji+=0x10) {
      for(int dan=1;dan<=5;dan++) {
         if(teban==SENTE){
                eval+=komaValue1[ban[suji+dan]];
         }else if(teban == GOTE){
        eval+=komaValue2[ban[suji+dan]];
      }
      }
    }
    // 次に、持ち駒
    for(int i=Koma.SFU;i<=Koma.STO;i++) {
      eval+=komaValue1[i]*hand[i];
    }
    for(int i=Koma.GFU;i<=Koma.GTO;i++) {
      eval+=komaValue2[i]*hand[i];
    }
  }

  public void initAll() {
    initEval();
    initKingPos();
  }

  // 局面を評価する関数。
```

```
public int evaluate() {
  return eval;
}

// CSA形式の棋譜ファイル文字列
static final String csaKomaTbl[] = {
  "   ","FU","KY","KE","GI","KI","KA","HI",
  "OU","TO","NY","NK ","NG","","UM","RY",
  ""  ,"+FU","+KY","+KE","+GI","+KI","+KA","+HI",
  "+OU","+TO","+NY","+NK","+NG",""   ,"+UM","+RY",
  ""  ,"-FU","-KY","-KE","-GI","-KI","-KA","-HI",
  "-OU","-TO","-NY","-NK","-NG",""   ,"-UM","-RY"
};



// 局面を表示用に文字列化
public String toString() {
  String s="";
  // 後手持ち駒表示
  s+="後手持ち駒：";
  for(int i=Koma.GFU;i<=Koma.GTO;i++) {
    if (hand[i]==1) {
      s+=Koma.toString(i);
    } else if (hand[i]>1) {
      s+=Koma.toString(i)+hand[i];
    }
  }
  s+="¥n";
  // 盤面表示
  s+=" 5  4  3  2  1¥n";
  s+="+---+---+---+---+---+¥n";
  for(int dan=1;dan<=5;dan++) {
    for(int suji=5;suji>=1;suji--) {
      s+="|";
      s+=Koma.toBanString(ban[(suji<<4)+dan]);
    }
```

```
      s+="|";
      s+=danStr[dan];
      s+="¥n";
      s+="+---+---+---+---+---+¥n";
    }
    // 先手持ち駒表示
    s+="先手持ち駒：";
    for(int i=Koma.SFU;i<=Koma.STO;i++) {
      if (hand[i]==1) {
        s+=Koma.toString(i);
      } else if (hand[i]>1) {
        s+=Koma.toString(i)+hand[i];
      }
    }
    s+="¥n";
    return s;
  }
}

package syougi6;
import java.util.Random;
import java.util.Vector;
import java.io.*;

import javax.swing.text.StyledEditorKit.ForegroundAction;

import org.omg.CORBA.IntHolder;

import syougi6.Koma;

public class Main implements Constants {
  // 初期盤面を与える
  static final int ShokiBanmen[][]={
        {Koma.GFU,Koma.GKI,Koma.GOU,Koma.GGI,Koma.GTO},
            {Koma.EMP,Koma.EMP,Koma.EMP,Koma.EMP,Koma.EMP},
            {Koma.EMP,Koma.EMP,Koma.EMP,Koma.EMP,Koma.EMP},
          {Koma.EMP,Koma.EMP,Koma.EMP,Koma.EMP,Koma.EMP},
          {Koma.STO,Koma.SGI,Koma.SOU,Koma.SKI,Koma.SFU},
```

```
  };

  // player[0]が先手が誰か、player[1]が後手が誰か。
  static Player player[]=new Player[2];

  static Vector kyokumenRireki=new Vector();

  // 使い方を表示する。
  static void usage() {
    System.out.println("使い方：");
    System.out.println("例：先手　人間　後手　コンピュータの場合");
    System.out.println("java jp.usapyonsoft.lesserpyon.Main HUMAN CPU");
    System.out.println("");
    System.out.println("初期局面を与えて対局開始することも可能です。");
    System.out.println("例：kyokumen.csaに初期局面が入っているとした場合");
    System.out.println("java jp.usapyonsoft.lesserpyon.Main HUMAN CPU
kyokumen.csa");
  }

  // メイン関数　引数は、先手番が誰か、後手番が誰か、初期局面は何か。
  // 誰か、は、人間ならば "HUMAN" CPUならば "CPU"を与える。
  // 初期局面が与えられなかった場合、平手の初期局面から。


  public static void main(String argv[]) {
    // 先手番、後手番が引数で与えられているかどうかチェック
//      for(int d = 0;d<=100;d++){
//           Vector kyokumenRireki = new Vector();
    if (argv.length<2) {
      // 引数が足りないようなら、使い方を表示して終わり。
      usage();
      return;
    }

    // 先手番が誰かを設定。
    if (argv[0].equals("HUMAN")) {
      player[0]=new Human();
    } else if (argv[0].equals("CPU")) {
```

```
  player[0]=new Sikou();
} else {
  // 引数がおかしいようなら、使い方を表示して終わり。
  usage();
  return;
}

// 後手番が誰かを設定。
if (argv[1].equals("HUMAN")) {
  player[1]=new Human();
} else if (argv[1].equals("CPU")) {
  player[1]=new Sikou();
} else {
  // 引数がおかしいようなら、使い方を表示して終わり。
  usage();
  return;
}

try {
  Kyokumen k=new Kyokumen();
  if (argv.length==2) {
    // 引数の指定がない場合、初期配置を使う。
    // 先手番
    k.teban=SENTE;
    for(int dan=1;dan<=5;dan++) {
      for(int suji=5;suji>=1;suji--) {
        k.ban[(suji<<4)+dan]=ShokiBanmen[dan-1][5-suji];
      }
    }
    // 諸々の初期化を行う。
    k.initAll();
  } else {
    // 引数で指定があった場合、CSA形式の棋譜ファイルを読み込む。
    String csaFileName=argv[2];
    File f=new File(csaFileName);
    BufferedReader in=new BufferedReader(new FileReader(f));
    Vector v=new Vector();
    String s;
```

```
      while((s=in.readLine())!=null) {
        System.out.println("Read:"+s);
        v.add(s);
      }
      String csaKifu[]=new String[v.size()];
      v.copyInto(csaKifu);
//        k.ReadCsaKifu(csaKifu);
      // ReadCsaKifuの中で必要な初期化が行われている。
    }

    int tesu=0;

    // 対戦のメインループ
    while(true) {
      tesu++;
      // 現在の局面を、局面の履歴に保存する。
      kyokumenRireki.add(k.clone());
      // 現在の局面での合法手を生成
      Vector v=GenerateMoves.generateLegalMoves(k);
      if (v.size()==0) {
        // 手番の側の負け
        if (k.teban==SENTE) {
          System.out.println("後手の勝ち！");
        } else {
          System.out.println("先手の勝ち！");
        }
        // 対局終了
        break;
      }
      // 千日手のチェック…連続王手の千日手には未対応。
      // 同一局面が何回出てきたか？
      int sameKyokumen=0;
      for(int i=0;i<kyokumenRireki.size();i++) {
        // 同一局面だったら…
        if (kyokumenRireki.elementAt(i).equals(k)) {
          // 同一局面の出てきた回数を増やす
          sameKyokumen++;
        }
```

```
        }
        if (sameKyokumen>=4) {
          // 同一局面４回以上の繰り返しなので、千日手。
          System.out.println("千日手です。");
          // 対局終了

          break;
        }

        // 局面を表示。
 //       System.out.println(k.toString());
        // ついでに、局面の評価値を表示


    //    System.out.println("現在の局面の評価値:"+k.evaluate());
        // 次の手を手番側のプレイヤーから取得
        Te te;
        if (k.teban==SENTE) {
          te=player[0].getNextTe(k);
        } else {
          te=player[1].getNextTe(k);
        }
        // 指された手を表示
//        System.out.println(te.toString());
        // 合法手でない手を指した場合、即負け
        if (!v.contains(te)) {
//          System.out.println("合法手でない手が指されました。");
          // 手番の側の負け
          if (k.teban==SENTE) {
            System.out.println("後手の勝ち！");
          } else if(k.teban==GOTE){
            System.out.println("先手の勝ち！");
          }else{
                System.out.println("引き分け");
          }
          // 対局終了
          break;
        }
```

```
        // 指された手で局面を進める。
        k.move(te);
        // moveでは、手番が変わらないので、局面の手番を変更する。
        if (k.teban==SENTE) {
          k.teban=GOTE;
        } else {
          k.teban=SENTE;
        }
      }
      // 対局終了。最後の局面を表示して、終わる。
 //     System.out.println("対局終了です。");
   //   System.out.println("最後の局面は…");
  //  System.out.println(k.toString());
    } catch(Exception ex) {
      ex.printStackTrace();
    }

  }
  }
//}

package syougi6;
import java.util.Random;
import java.util.Vector;

// コンピュータの思考ルーチン
public class Sikou implements Player,Constants {
  // ∞を表すための定数
  static final int INFINITE=99999999;

  // 読みの深さ
  static final int DEPTH_MAX=4;

  // 読みの最大深さ…これ以上の読みは絶対に不可能。
  static final int LIMIT_DEPTH=16;

  // αβカットを起こす関係で、ランダム着手は出来なくなる。
  // 詳細は解説にて。
```

```
// 最善手順を格納する配列
Te best[][]=new Te[LIMIT_DEPTH][LIMIT_DEPTH];

int leaf=0;
int node=0;


public Sikou() {

}

int negaMax(Te t,Kyokumen k,int alpha,int beta,int depth,int depthMax) {
  // 深さが最大深さに達していたらそこでの評価値を返して終了。
  if (depth>=depthMax) {
    leaf++;
    if (k.teban==SENTE) {
      return k.evaluate();
    } else {
      return -k.evaluate();
    }
  }
  node++;
  // 現在の局面での合法手を生成
  Vector v=GenerateMoves.generateLegalMoves(k);

  // 現在の指し手の候補手の評価値を入れる。
  int value=-INFINITE;

  // 合法手の中から、一手指してみて、一番よかった指し手を選択。
  for(int i=0;i<v.size();i++) {
    // 合法手を取り出す。
    Te te=(Te)v.elementAt(i);

    // その手で一手進める。
    k.move(te);
    // moveでは、先手後手を入れ替えないので…。
    if (k.teban==SENTE) {
```

```
  k.teban=GOTE;
} else {
  k.teban=SENTE;
}

// その局面の評価値を、さらに先読みして得る。
Te tmpTe=new Te(0,0,0,false,0);
int eval=-negaMax(tmpTe,k,-beta,-alpha,depth+1,depthMax);
k.back(te);
// backでは、先手後手を入れ替えないので…。
if (k.teban==SENTE) {
  k.teban=GOTE;
} else {
  k.teban=SENTE;
}

// 指した手で進めた局面が、今までよりもっと大きな値を返すか？
if (eval>value) {
  // 返す値を更新
  value=eval;
  // α値も更新
  if (eval>alpha) {
    alpha=eval;
  }
  // 最善手を更新
  best[depth][depth]=te;
  t.koma   =te.koma;
  t.from   =te.from;
  t.to     =te.to;
  t.promote=te.promote;
  // 最善手順を更新
  for(int j=depth+1;j<depthMax;j++) {
    best[depth][j]=best[depth+1][j];
  }
  // βカットの条件を満たしていたら、ループ終了。
  if (eval>=beta) {
    break;
  }
```

```
      }
    }
    return value;
  }

  public Te getNextTe(Kyokumen k) {
    leaf=node=0;

    // 投了にあたるような手で初期化。
    Te te=new Te(0,0,0,false,0);
    long time=System.currentTimeMillis();

    // 評価値最大の手を得る
    int v=negaMax(te,k,-INFINITE,INFINITE,0,DEPTH_MAX);

//    System.out.print("先読みの評価値:"+v);
//    System.out.print("  最善手順:");
    for(int i=0;i<DEPTH_MAX;i++) {
//      System.out.print(best[0][i]);
    }
//    System.out.println();

    time=System.currentTimeMillis()-time;
 //   System.out.println("leaf="+leaf+" node="+node+" time="+time+"ms");

    return te;
  }
}

package syougi6;
import java.util.Vector;
import java.io.*;

public class Human implements Player,Constants {
  // 一行入力用の読み込み元を用意しておく。
  // static メンバー変数で用意しておくのは、人間対人間の
  // 対戦時に、同じ読み込み元を使いたいため。
  // こうすることで、標準入力にファイルを使用できる。
```

```
static BufferedReader reader=
  new BufferedReader(new InputStreamReader(System.in));

public Te getNextTe(Kyokumen k) {
  // 現在の局面での合法手を生成
  Vector v=GenerateMoves.generateLegalMoves(k);
  // 返却する手の初期化…「投了」にあたるような、
  // 合法手でない手を生成しておく。
  Te te=new Te(0,0,0,false,0);

  do {
    if (k.teban==SENTE) {
    //  System.out.println("先手番です。");
    } else {
  //    System.out.println("後手番です。");
    }
//    System.out.println("指し手を入力して下さい。");
    // 一行入力
    String s="";
    try {
      s=reader.readLine();
    }catch(Exception e){
      // 読み込みエラー？
      e.printStackTrace();
      break;
    }
    // 入力された手が%TORYOだったら、投了して終わり。
    if (s.equals("%TORYO")) {
      break;
    }
    if (s.equals("p")) {
      // 合法手の一覧と局面を出力。
      for(int i=0;i<v.size();i++) {
        Te t=(Te)v.elementAt(i);
        System.out.println(t);
      }
      System.out.println(k);
      continue;
```

```
}
boolean promote=true;
if (s.length()==5) {
  if (s.substring(4,5).equals("*")) {
    // ５文字目が'*'だったら、『成り』
    promote=true;
  } else {
    // 何かおかしい…。
    System.out.println("入力が異常です。");
    // 局面を表示して、再入力を求める。
    System.out.println(k);
    continue;
  }
}
int fromSuji=0,fromDan=0,toSuji=0,toDan=0;
try {
  fromSuji=Integer.parseInt(s.substring(0,1));
  fromDan =Integer.parseInt(s.substring(1,2));
  toSuji  =Integer.parseInt(s.substring(2,3));
  toDan   =Integer.parseInt(s.substring(3,4));
}catch(Exception e){
  // 数値として読み込めなかったので、何か間違っている。
  System.out.println("手を読み込めませんでした。");
  System.out.println(""+fromSuji+""+fromDan+""+toSuji+""+toDan);
  // 局面を表示して、再入力を求める。
  System.out.println(k);
  continue;
}
// 駒
int koma=0;
// 最初の一桁が０の場合、駒打ち
if (fromSuji==0) {
  // この場合、二桁目に打つ駒の種類が入っている。
  // 駒は、手番の側の駒。
  koma=fromDan|k.teban;
  // fromDanをクリア。
  fromDan=0;
}
```

```
      int from=fromSuji*16+fromDan;
      int to  =toSuji  *16+toDan;
      if (fromSuji!=0) {
        koma=k.get(from);
      }
      te=new Te(koma,from,to,promote,k.get(to));
      if (!v.contains(te)) {
        // 合法手でないので、何か間違っている…。
        System.out.println(te);
        System.out.println("合法手ではありません。");
        // 局面を再表示。
        System.out.println(k);
      }
    } while(!v.contains(te));

    return te;
  }
}
```