卒業研究報告書

題目

将棋の駒の評価値と盤面サイズの考察

指導教員

石水 隆　講師

報告者

15-1-037-0043

西谷 昂真

近畿大学理工学部情報学科

平成 31 年 1 月 31 日提出

概要

５五将棋は通常の将棋を子供や将棋の初心者向けに通常の将棋盤を５×５盤に置き換え，駒の数も減らして分かり易くしたゲームである．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，動物将棋や５五将棋等のミニ将棋では適切な評価値は定まっていない．５五将棋は本将棋より盤面の大きさ，駒数等から本将棋に比べ，角，飛車の評価値が小さいことが考えられる．そこで本研究では，最適となる駒の評価値を求め，駒の評価値と盤面サイズの関係を検証する．

本研究では Java を用いて５五将棋 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，適切な駒の評価値を求める．また，盤面サイズを変更して同様の実験を行い，駒の評価値と盤面サイズの関係を調査する．

本研究で作成する将棋 AI は，NegaMax 法を用いて着手選択を行っている．余裕があれば駒同士の連携と玉の安全度の評価，機械学習など，ほかの手段についての着手選択についても考察したい．

本研究で作成した AI 同士の対戦を先手，後手共に評価値を変更しながら各 100 回行ったが，有意な結果は得られなかった．勝利数の結果が全て誤差の範囲に収まってしまい，最適な評価値であると言うには結果不足である．

目次

1. 序論

1.1 本研究の背景

コンピュータ将棋は，近年，着実に進化を続けており，対 人間戦ではコンピュータの勝利数が人間を上回ることも多くなってきた.

表 1にコンピュータ将棋の歴史を示す.

コンピュータ将棋の歴史は１９６８年に遡る．当時は近年のような人対 AI といった対戦ではなく，人間対コンピュータの詰め将棋早解き競争であった[1]．このプログラムを開発したのは株式会社日立製作所の越智利夫，亀井達弥，内ヶ崎儀一郎の 3 人で，プログラムは日立製作所が制作したコンピュータ HITAC5020E 上で動く，１９６７年でアマチュア初段の性能であった．１９７６年には初のコンピュータ将棋対局が行われ[2]，今日のコンピュータ将棋の幕開けとなった．この対局は，１９７６年９月３０日から１０月３日までの東京池袋の東武百貨店で情報化週間が催され，その席上ではじめての対局プログラムとプロ棋士である米長邦雄八段（当時）との対局が行われた．これは，公の場でのコンピュータとプロ棋士の初めての対戦であった．結果は４０手以内で終わったようで，『「残念ナガラ　形勢ハ　アナタニ有利デス　最後マデ　対局シタイノデスガ　コノママ続ケマスト　カナリ時間ガカカリソウデス　コノ辺デ　次ノ人ト交代シマショウ」』と途中でやめたことが記録に残っている[1].

表 1 コンピュータ将棋の歴史

年	出来事	文献
1968	人間対コンピュータの詰め将棋早解き競争が行なわれる	[1]
1976	初のコンピュータ将棋対局	[1], [2]
1990	第1回コンピュータ将棋選手権開催	[1]

1.2 ５五将棋

５五将棋とは，亜種の将棋ゲームの一種で，１９７０年頃に楠本茂信氏が作ったゲームと言われている[4]．プレー人口はそれほど多くはないが，大会が行われていたという記録もあることから，それ相応の難易度のゲームであることが知られている.

図 1 に５五将棋の初期配置図を示す．５五将棋は，本将棋を遊んだことがある人は，すぐに楽しめるゲームである．本将棋は，定跡や戦略を覚えるなど，初心者が越えなければならない壁があるが，５五将棋については，駒の動かし方を覚えてしまえば，誰でもすぐに楽しめるという手軽さがある.

1.3 コンピュータ将棋に関する既知の結果

本節では，コンピュータ将棋に関する既知の結果について述べる.

1.3.1　コンピュータ将棋の戦歴

これまでのコンピュータ将棋の歴戦として，１９９６年９月２０日のゲームプログラミングワークショップにおいて柿木将棋がコンピュータ将棋において初勝利を収めた[3].

近年では，２０１３年の第２回将棋電王戦から登場した ponanza が当該試合を含めた[3]. ponanza は２０１６年の第１期電王戦，２０１７年の第２期電王戦においてまで出場試合で，全て勝利を収めている.

表 2 に電王戦，表 3 に世界コンピュータ将棋選手権の結果を示す.

２０１２年，コンピュータ将棋とプロ棋士が戦う将棋電王戦が開催された．将棋電王戦は２０１５

年まで開催され，その後２０１６年，２０１７年に電王戦が開催された．対戦成績では，コンピュータ将棋がプロ棋士に勝ち越しており，コンピュータ将棋がプロ棋士を上回るくらい強くなったことがわかる．

一方，コンピュータ将棋同士の対戦では，１９９０年に始まった世界コンピュータ将棋選手権が開催されており，多くのコンピュータ将棋が出場している．

表 2 電王戦の結果 [3]

年	大会		人間		コンピュータ
2012	将棋電王戦	×	米長邦雄 永世棋聖	○	ボンクラーズ
2013	第2回将棋電王戦	×	三浦弘行 八段	○	GPS 将棋
		△	塚田泰明 九段	△	Puella α
		×	船江恒平 五段	○	ツツカナ
		×	佐藤慎一 四段	○	ponanza
		○	阿部光瑠 四段	×	習甦
2014	第3回将棋電王戦	×	屋敷伸之 九段	○	ponanza
		×	森下卓 九段	○	ツツカナ
		○	豊島将之 七段	×	YSS
		×	佐藤紳哉 六段	○	やねうら王
		×	菅井竜也 五段	○	習甦
2015	将棋電王戦 Final	○	阿久津主税 八段	×	AWAKE
		×	村山慈明 七段	○	ponanza
		×	稲葉陽 七段	○	やねうら王
		○	永瀬拓也 六段	×	Selene
		○	斎藤慎太郎 五段	×	Apery
2016	第1期電王戦	×	山崎隆之 八段	○	ponanza
		×	山崎隆之 八段	○	ponanza
2017	第2期電王戦	×	佐藤天彦 名人	○	ponanza
		×	佐藤天彦 名人	○	ponanza

表 3 世界コンピュータ将棋選手権の結果 [20]

回	年	優勝	準優勝	参加者数
1	1990	永世名人	柿木将棋	6
2	1991	森田将棋	極	9
3	1992	極	柿木将棋	10
4	1993	極	柿木将棋	14
5	1994	極	森田将棋	22
6	1996	金沢将棋	柿木将棋	25
7	1997	YSS	金沢将棋	33
8	1998	IS 将棋	金沢将棋	35
9	1999	金沢将棋	YSS	40
10	2000	IS 将棋	YSS	45
11	2001	IS 将棋	金沢将棋	55
12	2002	激指	IS 将棋	51
13	2003	IS 将棋	YSS	45
14	2004	YSS	激指	43
15	2005	激指	KCC 将棋	39
16	2006	Bonanza	YSS	43
17	2007	YSS	棚瀬将棋	40
18	2008	激指	棚瀬将棋	41
19	2009	GPS 将棋	大槻将棋	42
20	2010	激指	習甦	43
21	2011	ボンクラーズ	Bonanza	37
22	2012	GPS 将棋	Puella α	42
23	2013	Bonanza	ponanza	40
24	2014	Apery	ponanza	38
25	2015	ponanza	NineDayFever	39
26	2016	ponanza	技巧	51
27	2017	elmo	ponanza Chainer	50
28	2018	Hefeweizen	PAL	56

1.3.2 コンピュータ将棋の着手選択法

コンピュータ将棋が着手を決定するのに用いられている主な手法としては，局面の評価値計算，定跡データベースの利用，一定手数の先読み，終盤での詰み読み，モンテカルロ法，機械学習等がある．

局面の評価値計算とは，コンピュータ将棋で指し手を決定するときにこの指し手で進めた局面は何点，というように定めた，指し手選択の目安となる値（局面評価値）を用いる方法である．局面の評価値は，双方が持つ駒の価値，駒同士の連携度，駒の稼働範囲，玉の安全度などの要素を用いて決定される．この手法を用いる場合，評価値の計算に，どのような要素を用いるか，各要素にどれくらいの重みを付加するかが重要なポイントとなる．

定跡データベースとは，最善とされる決まった手順での指し方を集めたデータベースのことである．将棋には，プロ棋士が長年研究してきた定跡があり，また，プロ棋士の棋譜は公開されているので，そこから定跡データベースを構築できる．この手法を用いる場合，定跡に存在する局面では，プロ棋士が指した手を選択できるが，定跡に無い手，新手を出されると対応できない．

一定手数の先読みとは，一定数の先の手を読み，その手によって算出された評価値から，最も評価地が高い手を採用することである．先読みには，後述する MinMax 法やその改良である $\alpha\beta$ 法が用い

られる．一般に先読み手数が深くなるほど強くなるが，先読み手数が増えると探索時間が指数的に増えるため，適切な枝刈りを行わなければならない．どこまで読むのか，どのような条件で枝刈りするのかがこの手法を用いる場合のポイントとなる．

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，何手詰め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，ランダム法とも呼ばれ，大量の乱数を条件式に当てはめ，近似解を求める方法である．この手法を用いる場合，乱数で将棋の駒を動かし，何千局と対戦させ，最も勝率が高かった手を選ぶ．

機械学習とは，教師データであるさまざまな対戦データをもとに，最も効率の良い手をコンピュータに学習させる方法である．この手法を用いる場合，プロ棋士による大量の対戦データをコンピュータに学習させ，最善な手を選ぶ．

今回作成した AI では局面の評価値として盤面上の駒の価値を全部加算した上で，次に先手，後手の持ち駒の価値を全て加算することにより計算を行い，NegaMax 法を用いて一定手数先の局面において最も良い評価値を持つ局面となる手を選択することにより着手選択を行っている．

1.3.3　５五将棋に関する既知の結果

５五将棋においては，近年，参加者はそれ程多くはないが，大会が開催され，様々な国の選手も出場し，対戦を楽しんでいる．表 4 に UEC 杯５五将棋大会の歴史を示す．

表 4 UEC 杯５五将棋大会の歴史［16］

大会名称	開催日時	参加数	優勝	COM VS HUM
第 1 回 UEC 杯 5 五将棋大会	2007/11/25	14	55TACOS	○55TACOS VS 杉山卓弥
第 2 回 UEC 杯 5 五将棋大会	2008/12/7	12	COM:K55, KIDS: 三鷹二中, HUM: 山田剛	K55 VS ○山田剛, 55TACOS VS 山崎智博
第 3 回 UEC 杯 5 五将棋大会	2009/10/18	8	COM:K55, HUM: 山田剛	○K55 VS 山田剛
第 4 回 UEC 杯 5 五将棋大会	2010/12/4	7 (台湾 1)	COM:128 分の一里眼, HUM: 前田玄	
第 5 回 UEC 杯 5 五将棋大会	2011/12/4	12 (台湾 1，オランダ 1，アメリカ 1)	COM:128 分の一里眼, HUM: 前田玄	
第 6 回 UEC 杯 5 五将棋大会	2012/11/24	13 (中国 1，アメリカ 1，オランダ 1)	COM:128 分の一里眼, HUM: 前田玄	

また，５五将棋のソフトとして，iPhone アプリの ” ５五将棋 K55 ” (以下，K55) [5]が存在する．K55 は，柿木義一氏によって制作され，価格は 360 円である(2019 年 1 月 31 日 現在)．K55(PC 版)においては，2008 年と 2009 年のコンピュータ５五将棋大会で優勝している[16]．思考レベルは８段階であり，コンピュータとの対戦，および，1 台での人対人の対戦（棋譜入力）が可能である．

1.4 本研究の目的

本研究の目的として，強い５五将棋を作成することであり，それに伴い，まず駒の評価値と盤面サイズの関係について検証を行う．その過程において，５五将棋においての適切な駒の評価値を求める．

1.5 本研究の構成

本報告書の構成は，以下のとおりである．

2. ５五将棋
3. 駒の評価値を用いたコンピュータ将棋の着手選択法
4. ５五将棋プログラム
5. 将棋の駒の評価値と盤面サイズの検証
6. 実験結果および考察
7. 結論・今後の課題

2 5五将棋

本章では，本研究の対象である５五将棋について述べる．

５五将棋は，一般的な９×９盤将棋とは異なり，図 １に示す５×５盤で展開される将棋である．一般的な将棋の駒の動きと同一であるが，香車，桂馬を除いた駒の構成となっており，一般的な将棋の初期配列とは異なる．陣地は指し手の近傍１列となっており，相手の陣地に入ることによって成ることができる．また，本将棋と同じく打ち歩詰めは禁止となっている．

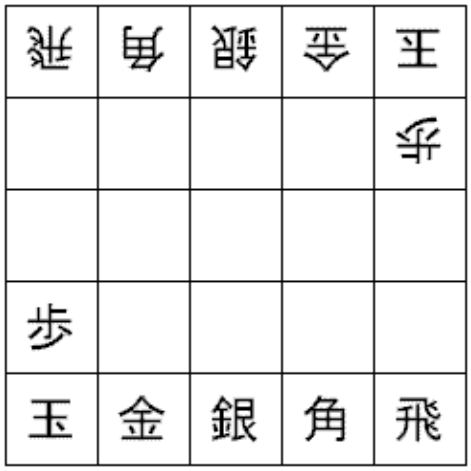

図 １ ５五将棋の初期配置図

3 駒の評価値を用いたコンピュータ将棋の着手選択法

今回作成した AI は，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の差を求めることにより局面の評価値を求めている．本将棋では，プロ棋士により妥当と思われる各駒の価値がいくつか示されている[1]．表 1 に本将棋の駒の評価値の例を示す．一方、５五将棋では駒の評価値は定まっておらず，適切な値を検証する必要がある．

本研究で作成した AI は，４手先の局面を求め，NegaMax 法にて最も良い評価値となる手を選択することで着手選択を行っている．

NagaMax 法とは MinMax 法の一種である．以下に MinMax 法ついて説明する．

1.ノードが自分の手番のときには、子ノードの中から最大の評価値のノードを選ぶ
　ノードの評価値は、選択した子ノードの評価値にする

2.ノードが相手の手番のときには、子ノードの中から最小の評価値のノードを選ぶ
ノードの評価値は、選択した子ノードの評価値にする

このアルゴリズムを MinMax 法（ミニマックス法）と呼ぶ[8]．図 2 に MinMax 法のアルゴリズムを示す．

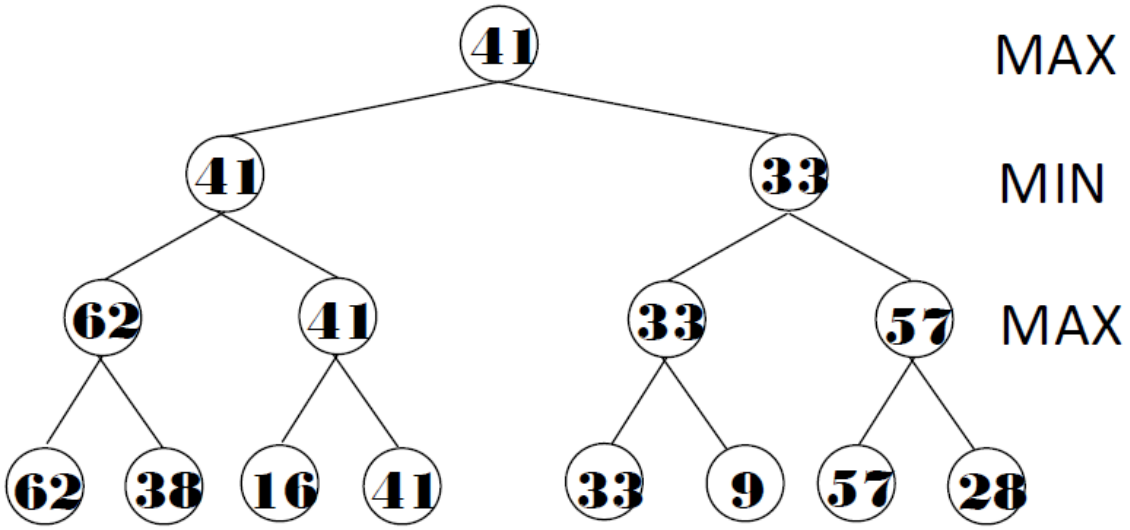

図 2 MinMax 法のアルゴリズム

次にα β法について説明する．
α β法は MinMax 法を改良したアルゴリズムである．MinMax 法と比較して探索ノード数を大幅に減らし、かつ MinMax 法と同じ結果を得られるという特徴がある．

NegaMax 法は以下のようなアルゴリズムで表すことが可能である．

$$Max(-Max(-Max(a,b),-Max(c,d),\dots),-Max(-Max(e,f),\dots),\dots)$$

NegaMax 法は，MinMax 法において，自分の手番か相手の手番化によって，最大化をするか最小化をするかが異なる点をどちらの手番でも最大化するようにしたものが NegaMax 法である．
また，局面の評価値の計算方法としては，盤面上の駒の価値を全部加算した上で，次に先手，後手の持ち駒の価値を全て加算することにより計算を行っている．

4　五将棋プログラム

本章では，本研究で作成した５五将棋 AI について述べる．
本研究では，Java を用いて５五将棋 AI を作成した．付録に本研究で作成した５五将棋 AI を示す．
本研究で作成した５五将棋 AI は，[1]の本将棋の将棋 AI を参考に，５五将棋用に改編を行ったものである．以下に本研究で作成した５五将棋ＡＩのプログラムの各クラスについて説明する．

4.1 Constants.java

Constaints.java は，インタフェースであり，各種定数の定義を行っている．例えば「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などである．
図 3 に Constants のクラス図を示す．

<<interface>> *Constants*
+ SENTE= 1<<4: int
+ GOTE= 1<<5: int
+ sujiStr[]: String
+ danStr[]: String

図 3 Constants のクラス図

4.2 GenerateMoves.java

GenerateMoves.java は，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動作を行うためのメソッド removeSelfMate，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・不成を判断しながら生成した手を追加するメソッド addTe，打ち歩詰めになっていなかどうかチェックする関数であり，相手の玉頭に歩を打つ場合，その手で一手進めてみて，相手の手番で GenerateLegalMove を行い，帰ってくる手がなかったなら打ち歩詰めになっていることを確認するメソッド isUtiFuDume，与えられた局面における合法手を生成するメソッド generateLegalMoves によって構成されている．**図 4** に GenerateMoves のクラス図を示す．

GenerateMoves
+ removeSelfMate (k : Kyokumen, v : Vector) : Vector
+ addTe (v: Vector, teban : int, koma : int, from : Position, to : Position) : void
+ isUtiFuDume (k : Kyokumen, te : Te) : boolean
+ generateLegalMoves (k : Kyokumen) : Vector

図 4 GenerateMoves のクラス図

4.3 Human.java

Human.java は，一行入力用の BufferedReader 型の reader をフィールドに持ち，現在の局面での次の手を生成，判定するメソッド getNextTe によって構成されている．図 5 に Human のクラス図を示す．

Human
~ reader: BufferdReader
+ getNextTe(Kyokumen: k): Te

図 5 Human のクラス図

4.4 Koma.java

Koma.java は，フィールドに駒の種類の定義，盤の外を表すための定数を持ち，先手の駒かどうか

の判定を行うメソッド isSente，後手の駒かどうかの判定を行うメソッド isGote，手番から見て自分の駒かどうか判定を行う isSelf，手番から見て相手の駒かどうか判定を行う isEnemy，駒の種類を取得するメソッド getKomashu，盤面の表示用に駒の文字列化を行うメソッド toBanString，持ち駒，手などの表示用の駒の文字列化を行うメソッド toString，駒が成れるかどうかを判定するメソッド canPromote によって構成されている．図 6 に Koma のクラス図を示す．

Koma
+ EMPTY=0: int
+ EMP=EMPTY: int
+ PROMOTE=8: int
+ FU=1: int
+ KY=2: int
+ KE=3: int
+ GI=4: int
+ KI=5: int
+ KA=6: int
+ HI=7: int
+ OU=8: int
+ TO=FU+PROMOTE: int
+ NY=KY+PROMOTE: int
+ NK=KE+PROMOTE: int
+ NG=GI+PROMOTE: int
+ UM=KA+PROMOTE: int
+ RY=HI+PROMOTE: int
+ SFU=SENTE+FU: int
+ SKY=SENTE+KY: int
+ SKE=SENTE+KE: int
+ SGI=SENTE+GI: int
+ SKI=SENTE+KI: int
+ SKA=SENTE+KA: int
+ SHI=SENTE+HI: int
+ SOU=SENTE+OU: int
+ STO=SENTE+TO: int
+ SNY=SENTE+NY: int
+ SNK=SENTE+NK: int
+ SNG=SENTE+NG: int
+ SUM=SENTE+UM: int
+ SRY=SENTE+RY: int
+ GFU=GOTE +FU: int
+ GKY=GOTE +KY: int
+ GKE=GOTE +KE: int
+ GGI=GOTE +GI: int
+ GKI=GOTE +KI: int
+ GKA=GOTE +KA: int
+ GHI=GOTE +HI: int
+ GOU=GOTE +OU: int

+	GTO=GOTE +TO: int
+	GNY=GOTE +NY: int
+	GNK=GOTE +NK: int
+	GNG=GOTE +NG: int
+	GUM=GOTE +UM: int
+	GRY=GOTE +RY: int
+	WALL=64: int
+	komaString[]: String
+	canPromote[]: boolean
+	isSente(koma: int): boolean
+	isGote(koma: int): boolean
+	isSelf(teban: int, koma: int): boolean
+	isEnemy(teban: int, koma: int): boolean
+	getKomashu(koma: int): int
+	toBanString(koma: int):String
+	toString(koma: int): String
+	canPromote(koma: int): boolean

図 6 Koma のクラス図

4.5 KomaMoves.java

KomaMoves.java は，インタフェースであり，動きの方向の定義を行っている．実装については，方向の定義に沿った，「段」の移動の定義をあらわした int 型の一次元配列 diffDan，同じく方向の定義に沿った，「筋」の移動の定義をあらわした int 型の一次元配列 diffSuji，ある方向にある駒が動けるかどうかを表すテーブルの boolean 型の２次元配列 canMove によって構成されている．図 7 に KomaMoves のクラス図を示す.

<<interface>> *KomaMoves*	
+	diffDan[]: int
+	diffSuji[]: int
+	canMove[][]: boolean

図 7 KomaMoves のクラス図

4.6 Kyokumen.java

Kyokumen.java は,局面の各種動作を表すクラスである.フィールドに盤面を表す２次元配列 ban，持ち駒を表す一次元配列 hand，手番を表す int 型の変数 teban，乱数生成用の randomValue や，先手の評価値を格納している一次元配列 komaValue_sente，後手の評価値を格納している一次元配列 komaValue_gote，CSA 形式の棋譜ファイル文字列を格納している一次元配列 csaKomaTb1 が存在する．メソッドについては，譜面のコピーを行うメソッド clone，局面が同一かどうか調べるメソッド equals，ある位置にある駒を取得するメソッド get，ある位置にある駒を置くメソッド put，駒を進めるメソッド move，玉を探して位置を返すメソッド searchGyoku，局面における評価関数を計算するメソッド evaluate，CSA 形式の棋譜ファイルから局面を読み込むメソッド ReadCsaKifu，局面を表示用に文字列化するメソッド，toString，乱数を生成するメソッド randomValue によって構成されて

いる．図 8 に Kyokumen のクラス図を示す.

Kyokumen	
+	ban[][]: int
+	hand[]: Vector
+	teban = SENTE: int
+	e: int
+	sum = 0: int
+	sum2 = 0: int
+	random = new Random(): Random
+	randomValue = random.nextInt(1000) – 500: int
~	komaValue_sente[]: int
~	komaValue_gote[]: int
~	csaKomaTb1[]: String
+	clone(): Object
+	equals(o: Object): boolean
+	equals(k: Kyokumen): boolean
+	get(p: Position): int
+	put(Position: p, koma: int): void
+	move(te: Te): void
+	searchGyoku(teban: int): Position
+	evaluate(): int
+	ReadCsaKifu(csaKifu: String[]): void
+	toString(): String
+	randomValue(): int

図 8 Kyokuen のクラス図

4.7 Main.java

Main.java は，フィールドに初期盤面を表す int 型の二次元配列 ShokiBanmen を持ち，player[0]，player[1]が誰であるかを表す Player 型の一次元配列 player，局面の動きを一つずつ格納する Vector 型の kyokumenRireki が存在する．メソッドについては，使い方を表示するメソッド usage，main メソッドによって構成されている．main メソッドは，考察のために将棋の AI について 100 回ループの実行，ループごとに局面の履歴を初期化，先手番の指定，対戦のメインループ，千日手の判定，評価値の表示，局面の表示等で構成されている．図 9 に Main のクラス図を示す.

Main	
~	ShokiBanmen[][]: int
~	Player[]= new Player[2]: Player
~	kyokumenRireki = new Vector(): Vector
~	Usage(): void
+	main(argv[]: String): void

図 9 Main のクラス図

4.8 Player.java

Player.java は，インタフェースである．図 10 に Player のクラス図を示す.

<<interface>> *Player*
~ getNextTe(k: Kyokumen): Te

図 10 Player のクラス図

4.9 Position.java

Position.java は，駒の位置を返すクラスである．フィールドに筋を表す int 型の変数 suji，段を表す int 型の変数 dan が存在する．メソッドについては，同一性比較用メソッド equals，コピーを返すメソッド clone, ある方向への動きを行うメソッド add, ある方向への逆向きの動きを行うメソッド sub によって構成されている．図 11 に Position のクラス図を示す.

Position
+ suji: int + dan: int
+ equals(p: Position): boolean + equals(p: Object): boolean + clone(): Object + add(diffSuji: int, diffDan: int): void + sub(diffSuji: int, diffDan: int): void + add(direct: int): void + sub(direct: int): void

図 11 Position のクラス図

4.10 Sikou.java

Sikou.java は，コンピュータの思考ルーチンをあらわすクラスである．フィールドに∞を表すための int 型の変数 INFINITE，読みの深さを表す int 型の変数 DEPTH_MAX，読みの最大の深さを表す int 型の変数 LIMIT_DEPTH，最善手順を格納する Te 型の２次元配列 best，negaMax 法を行う家庭で使用する int 型の変数 leaf，node が存在する．メソッドについては，negaMax 法のアルゴリズムを実際に行っているメソッド negaMax, 次の手を思考するメソッド getNextTe によって構成されている．図 12 に Sikou のクラス図を示す.

Sikou
~ INFINITE = 99999999: int ~ DEPTH_MAX=4: int ~ LIMIT_DEPTH=16: int ~ Best[][]=new Te[LIMIT_DEPTH][LIMIT_DEPTH]: Te ~ leaf=0: int

~	node=0: int
~	negaMax(t: Te, k: Kyokumen, alpha: int, beta: int, depth: int, depthMax: int): int
+	getNextTe(k: Kyokumen): Te

図 12 Sikou のクラス図

4.11 Te.java

Te.java は, 手を思考するクラスである. フィールドにどの駒が動いたかを表す int 型の変数 koma, 動く前の位置(持ち駒の場合、0 筋 0 段)を表す Position 型の変数 from, 動いた先の位置を表す Position 型の変数 to, 成る場合, true 成らない場合 false の値を判定する boolean 型の変数 promote が存在する. メソッドについては, 同一性比較用メソッド equals, 譜面のコピーを行うメソッド clone, 手を文字列で表現するメソッド toString によって構成されている. 図 13 に Te のクラス図を示す.

Te	
~	koma: int
~	from: Position
~	to: Position
~	promote: boolean
+	equals(te: Te): boolean
+	equals(_te: Object): boolean
+	clone(): Object
+	toString(): String

図 13 Te のクラス図

5 将棋の駒の評価値と盤面サイズの検証

本研究では 5 五将棋対戦 AI を作成し, 初期値である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, 最適な駒の評価値を求める. 本将棋の駒の評価値そのままでは, 盤面サイズが小さいことにより, 角や飛車の移動範囲が狭まることから, 検証結果に影響を与えると推測される. つまり, 本将棋に比べ, 角や飛車における重要度が下がるのではないかと考え, 角や飛車の評価値を下げることについて重点的に考察を行うことが求められるのではないかと考える.

6 実験結果および考察

先手, 後手それぞれの評価値を変化させながら考察を行った.

評価値を変化させ, 表 5 の評価値において, 角と飛車の評価値を変更し, 後手と対戦させた実行結果を表 6 に示す.

前章のとおり, 本将棋に比べ, 角や飛車における重要度が下がるのではないかと考えたことから, 角および飛車の評価値を初期値より重点的に増減させることにより考察を行った. 角および飛車の評価値を変更し, 実行を行った結果を表 6 に示す. 結果, 角, 飛車共に, 初期値±1000 程度であれば, 大きな差は見られなかったが, 角, 飛車共に評価値を 300, 500 にするなど極端に下げれば相手に非常に有利な結果となった. 誤差の範囲内ではあるが, 角および飛車の評価値をどちらも 5000 にした際, 最

も勝率が高い結果となった.

試行回数 100 回の場合，勝率 50%の対戦における危険度 95%の信頼区間に含まれる勝率は 40～60%となる．表 6 の結果は，後手の勝率を除き，全てこの信頼区間に含まれており，飛車角の評価値を変化させたことによる有意な勝率の差は見られなかった.

他にも，歩，銀，金，角，飛車など各駒において一つずつ評価値を変化させ，考察を行ってみたが，同じく有意な勝率の差は見られなかった．この結果は表 7～表 18 として付録に示す.

表 5 本将棋での駒の評価値の例【1】

歩	香	桂	銀	金	角	飛	玉
100	600	700	1000	1200	1800	2000	∞
と	杏	圭	全		馬	龍	
1200	1200	1200	1200		2000	2200	

表 6　対戦結果(試行回数各 100 回)

先手の駒の評価値				対戦結果			
角	馬	飛	龍	勝	負	分	勝率
2500	2500	2500	2500	48	48	4	50%
3600	4000	4000	4000	49	45	6	52%
3000	3000	3000	3000	41	51	8	45%
5000	5000	5000	5000	53	37	10	58%
5000	5000	5000	5000	48	38	14	55%
1800	2000	2000	2200	47	51	2	48%
1000	1000	1200	1200	47	49	4	49%
1000	1000	1200	1200	38	56	6	41%
1000	1000	1200	1200	39	51	10	43%
1000	1000	1300	1300	37	56	7	41%
1000	1000	1300	1300	38	58	4	40%
1000	1000	1400	1400	47	46	7	51%
1000	1000	1400	1400	52	43	5	55%
1000	1000	1500	1500	42	50	8	50%
1000	1000	1500	1500	50	46	4	52%
1300	1300	1200	1200	44	50	6	47%
1300	1300	1200	1200	49	48	3	51%
1000	1000	1000	1000	46	45	9	51%
1000	1000	1000	1000	49	44	7	53%
1200	1200	1200	1200	45	47	8	49%
1200	1200	1200	1200	39	53	8	43%
1200	1200	1300	1300	47	48	5	50%
1200	1200	1300	1300	40	57	3	42%
1200	1200	1400	1400	45	43	12	57%
1200	1200	1400	1400	38	57	5	41%
1200	1200	1500	1500	44	50	6	47%
1200	1200	1500	1500	48	46	6	51%

1200	1200	1600	1600	47	43	10	52%
1200	1200	1600	1600	42	50	8	46%
1200	1200	1700	1700	45	48	7	49%
1200	1200	1700	1700	46	48	6	49%
1300	1300	1200	1200	40	53	7	44%
1300	1300	1200	1200	35	57	8	39%
1300	1300	1300	1300	41	52	7	45%
1300	1300	1300	1300	52	45	3	54%
1300	1300	1400	1400	45	50	5	48%
1300	1300	1400	1400	43	52	5	46%
1300	1300	1500	1500	47	50	3	49%
1300	1300	1500	1500	51	42	7	55%
1400	1400	1200	1200	47	48	5	50%
1400	1400	1200	1200	56	36	8	60%
1400	1400	1300	1300	47	50	3	49%
1400	1400	1300	1300	54	38	8	58%
1400	1400	1400	1400	47	47	6	50%
1400	1400	1400	1400	48	45	7	52%
1400	1400	1500	1500	52	43	5	55%
1400	1400	1500	1500	50	44	6	53%
1500	1500	1200	1200	41	51	8	45%
1500	1500	1200	1200	49	46	5	52%
1500	1500	1300	1300	44	51	5	47%
1500	1500	1300	1300	42	52	6	45%
1500	1500	1400	1400	51	44	5	54%
1500	1500	1400	1400	42	49	9	47%
1500	1500	1500	1500	53	40	7	57%
1500	1500	1500	1500	48	50	2	49%
800	1000	1000	1200	49	49	2	50%
800	1000	1000	1200	50	49	1	51%
500	500	800	800	39	55	6	42%
500	500	500	500	38	57	5	41%
300	300	300	300	26	70	4	28%
300	300	300	300	27	69	4	29%

7　結論・今後の課題

本将棋と比較し，５五将棋は盤面サイズが５×５と小さいため，本将棋と比べ，角や飛車の評価値が小さいと予想していたが，実験結果では各駒において有意な差は見られなかった．今後の課題として，今回は駒の評価値のみに基づいて着手選択を行っているが，先述のように駒同士の連携と玉の安全度の評価，機械学習など，ほかの手段についての着手選択を行った際，どのような結果となるのか調査を行いたいと考えている．

謝辞

本研究を行うにあたり，ご指導いただきました石水隆講師には，研究を行うにあたり，適切で丁寧な意見やご指導，励ましの言葉をいただき，大変お世話になりました．また，一緒に研究に励んできた研究室のメンバーにも支えていただき，この場を借りて深く感謝するとともに厚く御礼申し上げます．ありがとうございました．

参考文献

[1] 清 慎一, コンピュータ将棋の初期の歴史, 情報処理学会 研究報告, Vol.2014-GI-31, pp.1-8, (2014) http://id.nii.ac.jp/1001/00099265/

[2] 高田淳一, コンピュータ将棋の歴史, http://www.junichi-takada.jp/computer_shogi/history.html

[3] 高田淳一, コンピュータ将棋 対 人間 対戦の記録, http://www.junichi-takada.jp/computer_shogi/comvshuman.html

[4] ５五将棋のルール, 伊藤毅志研究室, 電気通信大学, (2012) http://minerva.cs.uec.ac.jp/~uec55shogi/wiki.cgi?page=5%B8%DE%BE%AD%B4%FD%A4%CE%A5%EB%A1%BC%A5%EB

[5] ５五将棋 K55 for iPhone, 4Gamer.net https://www.4gamer.net/games/080/G008072/

[6] 池 泰弘 : Java 将棋のアルゴリズム, 工学社 (2007)

[7] 伊藤毅志, 新沢剛：モンテカルロ法を用いた５五将棋システム, 情報処理学会 研究報告 2007-GI-18, pp.1-6 (2007) http://id.nii.ac.jp/1001/00058488/

[8] Daiki Sanno, リバーシプログラムの作り方, 第２章 ゲーム木と探索 http://www.es-cube.net/es-cube/reversi/sample/html/2_4.html

[9] 高野大輔, 万小紅, 田中啓治, 伊藤毅志, ５五将棋の学習における認知過程の変化, 情報処理学会研究報告 Vol.2011-GI-26 No.8, pp.1-8 (2011) http://id.nii.ac.jp/1001/00074627/

[10] 伊藤毅志, 第 1 回 UEC 杯５五将棋大会報告(2007 年 11 月), 情報処理学会 研究報告, ゲーム情報学研究会, Vol.2008-GI-019, pp. 9-16, (2008), http://id.nii.ac.jp/1001/00058480/

[11] 伊藤毅志, 第 2 回 UEC 杯５五将棋大会報告(2008 年 12 月), 情報処理学会 研究報告, ゲーム情報学研究会、Vol.2009-GI-21, pp. 1-8, (2009), http://id.nii.ac.jp/1001/00061653/

[12] 伊藤毅志, 第 3 回 UEC 杯５五将棋大会報告(2009 年 11 月), 情報処理学会 研究報告, ゲーム情報学研究会, Vol.2010-GI-23, No.9, pp. 1-7, (2010), http://id.nii.ac.jp/1001/00068069/

[13] 伊藤毅志, ５五将棋大会 2010 報告, 情報処理学会 研究報告, ゲーム情報学研究会, Vol.2011-GI26, No.6, pp.1-7 (2011), http://id.nii.ac.jp/1001/00074625/

[14] 伊藤毅志, ５五将棋大会 2011 年の動向, 情報処理学会 研究報告, ゲーム情報学ケ久回, Vol.2012-GI-28, No.2, pp.1-6, (2012), http://id.nii.ac.jp/1001/00082809/

[15] 伊藤毅志, ５五将棋大会 2012 年の動向、情報処理学会 研究報告, ゲーム情報学研究会, 2013-GI-30, No.2, pp.1-6 (2013), http://id.nii.ac.jp/1001/00092707/

[16] 伊藤毅志, ５五将棋大会の動向(2013 年～2014 年), 情報処理学会 研究報告 Vol.2015-GI-33 No.1, pp.1-5 (2015), http://id.nii.ac.jp/1001/00113628/

[17] 中西啓, 将棋対決・第 1 回 チェスコンピュータからの応用でもアマ 20 級, 人間 vs コンピュータ 10 番勝負！, HH News & Report, 2013 年 8 月 8 日, https://www.hummingheads.co.jp/reports/series/ser03/130808.html

[18] 中西啓, 将棋対決・第 2 回 しのぎを削る将棋ソフトと、Bonanza の登場, 人間 vs コンピュータ 10 番勝負！, HH News & Reports, 2013 年 9 月 12 日, https://www.hummingheads.co.jp/reports/series/ser03/130912.html

[19] 中西啓, 将棋対決・第 3 回 人間を超えたかに見える、将棋ソフトの弱点, 人間 vs コンピュータ 10 番勝負！, HH News & Reports, 2013 年 10 月 28 日, https://www.hummingheads.co.jp/reports/series/ser03/131028.html

[20] コンピュータ将棋協会 , 世界コンピュータ将棋選手権 http://www2.computer-shogi.org/wcsc/

付録について

付録A　すべての対戦結果

表 7　対戦結果(試行回数各 100 回)

先手の駒の評価値		対戦結果			
銀	全	勝	負	分	勝率
5000	5000	48	38	14	55%
5000	5000	42	50	8	46%
3000	3000	49	42	9	54%
3000	3000	46	47	7	50%
0	0	38	57	5	41%

表 8　対戦結果(試行回数各 100 回)

後手の駒の評価値		対戦結果			
銀	全	勝	負	分	勝率
0	0	55	37	8	41%

表 9　対戦結果(試行回数各 100 回)

先手の駒の評価値		対戦結果			
飛	竜	勝	負	分	勝率
9000	9000	44	53	3	46%
8000	8000	44	51	5	47%
7000	7000	39	51	10	44%
7000	7000	45	52	3	47%
5000	5200	45	50	5	48%
5000	5000	54	39	7	58%
5000	5000	47	51	2	48%
1000	1200	46	45	9	51%
100	100	25	68	7	29%
100	100	20	72	8	24%

表 10　対戦結果(試行回数各 100 回)

後手の駒の評価値		対戦結果			
飛	竜	勝	負	分	勝率
1000	1200	54	40	6	43%

表 11　対戦結果(試行回数各 100 回)

先手の駒の評価値		対戦結果			
角	馬	勝	負	分	勝率
5000	5000	48	44	8	52%
800	1000	55	40	5	58%
100	100	14	73	13	21%
100	100	17	77	6	20%
100	100	30	65	5	33%
100	100	38	55	7	42%

表 12　対戦結果(試行回数各 100 回)

後手の駒の評価値		対戦結果			
角	馬	勝	負	分	勝率
800	1000	52	39	9	44%
100	100	87	8	5	11%
100	100	81	10	9	15%

表 13　対戦結果(試行回数各 100 回)

先手の駒の評価値	対戦結果			
金	勝	負	分	勝率
3000	43	48	9	48%
3000	33	56	11	39%
200	27	65	8	31%

表 14　対戦結果(試行回数各 100 回)

後手の駒の評価値	対戦結果			
金	勝	負	分	勝率
3000	62	29	9	34%
3000	61	30	9	35%
200	67	27	6	30%

表 15　対戦結果(試行回数各 100 回)

先手の駒の評価値		対戦結果			
歩	と	勝	負	分	勝率
−900	200	26	70	4	28%
2000	2000	33	60	7	37%
2000	2000	29	55	16	37%

表 16　対戦結果(試行回数各 100 回)

後手の駒の評価値		対戦結果			
歩	と	勝	負	分	勝率
−900	200	61	32	7	36%
2000	2000	40	48	12	54%
2000	2000	40	55	5	58%

表 17　対戦結果(試行回数各 100 回)

先手の駒の評価値					対戦結果			
玉	角	馬	飛	龍	勝	負	分	勝率
15000	8000	8000	8000	8000	58	35	7	62%
15000	8000	8000	8000	8000	46	43	11	52%

表 18　対戦結果(試行回数各 100 回)

先手の駒の評価値			対戦結果			
金	銀	全	勝	負	分	勝率
2000	2000	2000	48	46	6	51%
2000	2000	2000	50	49	1	51%

付録Ｂ　ソースプログラム

＜Constants.java＞

```
 1 package jp.usapyonsoft.lesserpyon;
 2
 3 // 各種定数の定義
 4 public interface Constants {
 5   // 「先手」の定義
 6   public final static int SENTE=1<<4;
 7   // 「後手」の定義
 8   public final static int GOTE =1<<5;
 9
10   // 筋を表す文字列の定義
11   public final static String sujiStr[]={
12 //    "","１","２","３","４","５","６","７","８","９"
13     "","１","２","３","４","５"
14   };
15
16   // 段を表す文字列の定義
17   public final static String danStr[]={
18 //    "","一","二","三","四","五","六","七","八","九"
19     "","一","二","三","四","五"
20   };
21
22
23 }
```

＜GenerateMoves.java＞

```
 1 package jp.usapyonsoft.lesserpyon;
 2 import java.util.Vector;
 3
 4 public class GenerateMoves implements Constants,KomaMoves {
 5
 6   // 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、
 7   // 王手がかかっている手は取り除く。
 8   public static Vector removeSelfMate(Kyokumen k,Vector v) {
 9     Vector removed=new Vector();
10     for(int i=0;i<v.size();i++) {
11       // 手を取り出す。
12       Te te=(Te)v.elementAt(i);
13
14       // その手で１手進めてみる
15       Kyokumen test=(Kyokumen)k.clone();
16       test.move(te);
17
18       // 自玉を探す
19       Position gyokuPosition=test.searchGyoku(k.teban);
20
```

```
21        // 王手放置しているかどうかフラグ
22        boolean isOuteHouchi=false;
23
24        // 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
25        for(int direct=0;direct<12 && !isOuteHouchi;direct++) {
26          // 方向の反対方向にある駒を取得
27          Position pos=(Position)gyokuPosition.clone();
28          pos.sub(direct);
29          int koma=test.get(pos);
30          // その駒が敵の駒で、玉方向に動けるか？
31          if (Koma.isEnemy(test.teban,koma) && canMove[direct][koma]) {
32            // 動けるなら、この手は王手を放置しているので、
33            // この手は、removed に追加しない。
34            isOuteHouchi=true;
35            break;
36          }
37        }
38
39        // 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
40        for(int direct=0;direct<8 && !isOuteHouchi;direct++) {
41          // 方向の反対方向にある駒を取得
42          Position pos=(Position)gyokuPosition.clone();
43          int koma;
44          // その方向にマスが空いている限り、駒を探す
45          for(pos.sub(direct),koma=test.get(pos);
46           koma!=Koma.WALL;pos.sub(direct),koma=test.get(pos)) {
47            // 味方駒で利きが遮られているなら、チェック終わり。
48            if (Koma.isSelf(test.teban,koma)) break;
49            // 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。
50            if (Koma.isEnemy(test.teban,koma) && canJump[direct][koma]) {
51              isOuteHouchi=true;
52              break;
53            }
54            // 敵駒で利きが遮られているから、チェック終わり。
55            if (Koma.isEnemy(test.teban,koma)) {
56              break;
57            }
58          }
59        }
60        if (!isOuteHouchi) {
61          removed.add(te);
62        }
63      }
64      return removed;
65    }
66
67    // 与えられた Vector に、手番、駒の種類、移動元、移動先を考慮して、
68    // 成る・不成りを判断しながら生成した手を追加する。
```

```
69   public static void addTe(Vector v,int teban,int koma,Position from,Position to) {
70     if (teban==SENTE) {
71       // 先手番
72 if ((Koma.getKomashu(koma)==Koma.KY || Koma.getKomashu(koma)==Koma.FU) && to.dan==1) {
73         // 香車か歩が１段目に進むときには、成ることしか選べない。
74         Te te=new Te(koma,from,to,true);
75         v.add(te);
76       } else if (Koma.getKomashu(koma)==Koma.KE && to.dan<=1) {
77           // 桂馬が１段目以上に進む時には、成ることしか選べない。
78         Te te=new Te(koma,from,to,true);
79         v.add(te);
80       } else if ((to.dan<=1 || from.dan<=1) && Koma.canPromote(koma)) {
81         // 駒の居た位置が相手陣か、進む位置が相手陣で、
82         // 駒が成ることが出来るなら
83         // 成りと不成りの両方の手を生成
84         Te te=new Te(koma,from,to,true);
85         v.add(te);
86         te=new Te(koma,from,to,false);
87         v.add(te);
88       } else {
89         // 不成りの手のみ生成
90         Te te=new Te(koma,from,to,false);
91         v.add(te);
92       }
93     } else {
94       // 後手番
95 if ((Koma.getKomashu(koma)==Koma.KY || Koma.getKomashu(koma)==Koma.FU) && to.dan==5) {
96         // 香車か歩が五段目に進むときには、成ることしか選べない。
97         Te te=new Te(koma,from,to,true);
98         v.add(te);
99       }
100 //      else if (Koma.getKomashu(koma)==Koma.KE && to.dan>=8) {
101 //        // 桂馬が八段目以上に進む時には、成ることしか選べない。
102 //        Te te=new Te(koma,from,to,true);
103 //        v.add(te);
104 //      }
105       else if ((to.dan>=5 || from.dan>=5) && Koma.canPromote(koma)) {
106         // 駒の居た位置が相手陣か、進む位置が相手陣で、
107         // 駒が成ることが出来るなら
108         // 成りと不成りの両方の手を生成
109         Te te=new Te(koma,from,to,true);
110         v.add(te);
111         te=new Te(koma,from,to,false);
112         v.add(te);
113       } else {
114         // 不成りの手のみ生成
115         Te te=new Te(koma,from,to,false);
116         v.add(te);
```

```
117       }
118     }
119   }
120
121   // 打ち歩詰めになっていないかどうかチェックする関数
122   // 相手の玉頭に歩を打つ場合、
123   // その手で一手進めてみて、相手の手番で GenerateLegalMove を行い、
124   // 帰ってくる手がなかったなら打ち歩詰めになっている。
125   public static boolean isUtiFuDume(Kyokumen k,Te te) {
126     if (te.from.suji!=0 && te.from.dan!=0) {
127       // 駒を打つ手ではないので、打ち歩詰めではない。
128       return false;
129     }
130     if (Koma.getKomashu(te.koma)!=Koma.FU) {
131       // 歩を打つ手ではないので、打ち歩詰めではない。
132       return false;
133     }
134     int teban;
135     int tebanAite;
136     if ((te.koma&SENTE)!=0) {
137       // 先手の歩を打つから、自分の手番は先手、相手の手番は後手
138       teban=SENTE;
139       tebanAite=GOTE;
140     } else {
141       // そうでない時は、自分の手番は後手、相手の手番は先手
142       teban=GOTE;
143       tebanAite=SENTE;
144     }
145     Position gyokuPositionAite=k.searchGyoku(tebanAite);
146     if (teban==SENTE) {
147       if (gyokuPositionAite.suji!=te.to.suji || gyokuPositionAite.dan!=te.to.dan-1) {
148         // 相手の玉の頭に歩を打つ手ではないので、打ち歩詰めになっていない。
149         return false;
150       }
151     } else {
152       if (gyokuPositionAite.suji!=te.to.suji || gyokuPositionAite.dan!=te.to.dan+1) {
153         // 相手の玉の頭に歩を打つ手ではないので、打ち歩詰めになっていない。
154         return false;
155       }
156     }
157     // 実際に一手進めてみる…。
158     Kyokumen test=(Kyokumen)k.clone();
159     test.move(te);
160     test.teban=tebanAite;
161     // その局面で、相手に合法手があるか？なければ、打ち歩詰め。
162     Vector v=generateLegalMoves(test);
163     if (v.size()==0) {
164       // 合法手がないので、打ち歩詰め。
```

```
165       return true;
166     }
167     return false;
168   }
169
170   // 与えられた局面における合法手を生成する。
171   public static Vector generateLegalMoves(Kyokumen k) {
172     Vector v=new Vector();
173
174     // 盤上の手番の側の駒を動かす手を生成
175     for(int suji=1;suji<=5;suji++) {
176       for(int dan=1;dan<=5;dan++) {
177         Position from=new Position(suji,dan);
178         int koma=k.get(from);
179         // 自分の駒であるかどうか確認
180         if (Koma.isSelf(k.teban,koma)) {
181           // 各方向に移動する手を生成
182           for(int direct=0;direct<12;direct++) {
183             if (canMove[direct][koma]) {
184               // 移動先を生成
185               Position to=new Position(suji+diffSuji[direct],dan+diffDan[direct]);
186               // 移動先は盤内か？
187               if (1<=to.suji && to.suji<=5 && 1<=to.dan && to.dan<=5) {
188                 // 移動先に自分の駒がないか？
189                 if (Koma.isSelf(k.teban,k.get(to))) {
190                   // 自分の駒だったら、次の方向を検討
191                   continue;
192                 }
193                 // 成る・不成りを考慮しながら、手をｖに追加
194                 addTe(v,k.teban,koma,from,to);
195               }
196             }
197           }
198           // 各方向に「飛ぶ」手を生成
199           for(int direct=0;direct<8;direct++) {
200             if (canJump[direct][koma]) {
201               // そちら方向に飛ぶことが出来る
202               for(int i=1;i<5;i++) {
203                 // 移動先を生成
204                Position to=new Position(suji+diffSuji[direct]*i,dan+diffDan[direct]*i);
205                 // 行き先が盤外だったら、そこには行けない
206                 if (k.get(to)==Koma.WALL) break;
207                 // 行き先に自分の駒があったら、そこには行けない
208                 if (Koma.isSelf(k.teban,k.get(to))) break;
209                 // 成る・不成りを考慮しながら、手をｖに追加
210                 addTe(v,k.teban,koma,from,to);
211                 // 空き升でなければ、ここで終わり
212                 if (k.get(to)!=Koma.EMPTY) break;
```

```
213                 }
214               }
215             }
216           }
217         }
218       }
219
220
221       // 手番の側の駒を打つ手を生成
222
223       // 手番の側の持ち駒で、その駒を既に打ったかどうかチェックするための配列
224       // 何もなし、歩〜飛車まで
225       boolean isPutted[]={false,false,false,false,false,false,false,false};
226
227       // 手番の側の持ち駒
228       Vector motigoma;
229       if (k.teban==SENTE) {
230         motigoma=k.hand[0];
231       } else {
232         motigoma=k.hand[1];
233       }
234
235       // まず、手番の側の持ち駒でループ
236       for(int i=0;i<motigoma.size();i++) {
237         // 持ち駒を一つ取り出す
238         int koma=((Integer)motigoma.elementAt(i)).intValue();
239         // 駒の種類を得る
240         int komashu=Koma.getKomashu(koma);
241         if (isPutted[komashu]) {
242           // 既にその駒を打ったことがあるなら、同じ駒を打つ手を生成するのは
243           // 無駄になるので、行わない。
244           continue;
245         }
246         // この駒を打ったことがある、と印を付ける
247         isPutted[komashu]=true;
248         // 盤面の各升目でループ
249         for(int suji=1;suji<=5;suji++) {
250           // 二歩にならないかどうかチェック
251           if (komashu==Koma.FU) {
252             // 二歩のチェック用変数
253             boolean isNifu=false;
254             // 二歩チェック
255             // 同じ筋に、手番の側の歩がいないことを確認する
256             for(int dan=1;dan<=5;dan++) {
257               Position p=new Position(suji,dan);
258               // 手番の側の歩が、同じ筋にいないかどうかチェックする
259               if (k.get(p)==(k.teban|Koma.FU)) {
260                 // 二歩になっている。
```

```
261             isNifu=true;
262             break;
263           }
264         }
265         if (isNifu) {
266           // 二歩になっているので、打つ手を生成しない。
267           // 次の筋へ
268           continue;
269         }
270       }
271       for(int dan=1;dan<=5;dan++) {
272         // 駒が桂馬の場合の扱い
273         if (komashu==Koma.KE) {
274           if (k.teban==SENTE && dan<=2) {
275             // 先手なら、二段目より上に桂馬は打てない
276             continue;
277           } else if (k.teban==GOTE && dan>=5) {
278             // 後手なら、八段目より下に桂馬は打てない
279             continue;
280           }
281         }
282         // 駒が歩、または香車の場合の扱い
283         if (komashu==Koma.FU || komashu==Koma.KY) {
284           if (k.teban==SENTE && dan==1) {
285             // 先手なら、一段目に歩と香車は打てない
286             continue;
287           } else if (k.teban==GOTE && dan==5) {
288             // 後手なら、九段目に歩と香車は打てない
289             continue;
290           }
291         }
292         // 移動元…駒を打つ手は、0,0
293         Position from=new Position(0,0);
294         // 移動先、駒を打つ場所
295         Position to=new Position(suji,dan);
296
297         // 空き升でなければ、打つ事は出来ない。
298         if (k.get(to)!=Koma.EMPTY) {
299           continue;
300         }
301         // 手の生成…駒を打つ際には、常に不成である。
302         Te te=new Te(koma,from,to,false);
303         // 打ち歩詰めの特殊扱い
304         if (isUtiFuDume(k,te)) {
305           // 打ち歩詰めなら、そこに歩は打てない
306           continue;
307         }
308         // 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。
```

```
309             v.add(te);
310           }
311         }
312       }
313
314       // 生成した各手について、指してみて
315       // 自分の玉に王手がかかっていないかどうかチェックし、
316       // 王手がかかっている手は取り除く。
317       v=removeSelfMate(k,v);
318
319       return v;
320     }
321 }
```

＜Human.java＞

```
  1 package jp.usapyonsoft.lesserpyon;
  2 import java.util.Vector;
  3 import java.io.*;
  4
  5 public class Human implements Player,Constants {
  6   // 一行入力用の読み込み元を用意しておく。
  7   // static メンバー変数で用意しておくのは、人間対人間の
  8   // 対戦時に、同じ読み込み元を使いたいため。
  9   // こうすることで、標準入力にファイルを使用できる。
 10   static BufferedReader reader=
 11     new BufferedReader(new InputStreamReader(System.in));
 12
 13   public Te getNextTe(Kyokumen k) {
 14     // 現在の局面での合法手を生成
 15     Vector v=GenerateMoves.generateLegalMoves(k);
 16     // 返却する手の初期化…「投了」にあたるような、
 17     // 合法手でない手を生成しておく。
 18     Position toryoPos=new Position(0,0);
 19     Te te=new Te(0,toryoPos,toryoPos,false);
 20
 21     do {
 22       if (k.teban==SENTE) {
 23         System.out.println("先手番です。");
 24       } else {
 25         System.out.println("後手番です。");
 26       }
 27       System.out.println("指し手を入力して下さい。");
 28       // 一行入力
 29       String s="";
 30       try {
 31         s=reader.readLine();
 32       }catch(Exception e){
 33         // 読み込みエラー？
```

```
34          e.printStackTrace();
35          break;
36        }
37        // 入力された手が%TORYOだったら、投了して終わり。
38        if (s.equals("%TORYO")) {
39          break;
40        }
41        if (s.equals("p")) {
42          // 合法手の一覧と局面を出力。
43          for(int i=0;i<v.size();i++) {
44            Te t=(Te)v.elementAt(i);
45            System.out.println(t);
46          }
47          System.out.println(k);
48          continue;
49        }
50        boolean promote=false;
51        if (s.length()==5) {
52          if (s.substring(4,5).equals("*")) {
53            // ５文字目が'*'だったら、『成り』
54            promote=true;
55          } else {
56            // 何かおかしい…。
57            System.out.println("入力が異常です。");
58            // 局面を表示して、再入力を求める。
59            System.out.println(k);
60            continue;
61          }
62        }
63        int fromSuji=0,fromDan=0,toSuji=0,toDan=0;
64        try {
65          fromSuji=Integer.parseInt(s.substring(0,1));
66          fromDan =Integer.parseInt(s.substring(1,2));
67          toSuji  =Integer.parseInt(s.substring(2,3));
68          toDan   =Integer.parseInt(s.substring(3,4));
69        }catch(Exception e){
70          // 数値として読み込めなかったので、何か間違っている。
71          System.out.println("手を読み込めませんでした。");
72          System.out.println(""+fromSuji+""+fromDan+""+toSuji+""+toDan);
73          // 局面を表示して、再入力を求める。
74          System.out.println(k);
75          continue;
76        }
77        // 駒
78        int koma=0;
79        // 最初の一桁が０の場合、駒打ち
80        if (fromSuji==0) {
81          // この場合、二桁目に打つ駒の種類が入っている。
```

```
 82          // 駒は、手番の側の駒。
 83          koma=fromDan|k.teban;
 84          // fromDan をクリア。
 85          fromDan=0;
 86        }
 87        Position from=new Position(fromSuji,fromDan);
 88        Position to  =new Position(toSuji,  toDan);
 89        if (fromSuji!=0) {
 90          koma=k.get(from);
 91        }
 92        te=new Te(koma,from,to,promote);
 93        if (!v.contains(te)) {
 94          // 合法手でないので、何か間違っている…。
 95          System.out.println(te);
 96          System.out.println("合法手ではありません。");
 97          // 局面を再表示。
 98          System.out.println(k);
 99        }
100      } while(!v.contains(te));
101
102      return te;
103    }
104 }
```

＜Koma.java＞

```
  1 package jp.usapyonsoft.lesserpyon;
  2
  3 // 駒
  4 class Koma implements Constants,Cloneable {
  5   // 駒の種類の定義
  6   public static final int EMPTY=0;          // 「空」
  7   public static final int EMP=EMPTY;        // 「空」の別名。
  8   public static final int PROMOTE=8;        // 「成り」フラグ
  9
 10   public static final int FU= 1;           // 「歩」
 11   public static final int KY= 2;           // 「香車」
 12   public static final int KE= 3;           // 「桂馬」
 13   public static final int GI= 4;           // 「銀」
 14   public static final int KI= 5;           // 「金」
 15   public static final int KA= 6;           // 「角」
 16   public static final int HI= 7;           // 「飛車」
 17   public static final int OU= 8;           // 「玉将」
 18   public static final int TO=FU+PROMOTE;   // 「と金」＝「歩」＋成り
 19   public static final int NY=KY+PROMOTE;   // 「成り香」＝「香車」＋成り
 20   public static final int NK=KE+PROMOTE;   // 「成り桂」＝「桂馬」＋成り
 21   public static final int NG=GI+PROMOTE;   // 「成り銀」＝「銀」＋成り
 22   public static final int UM=KA+PROMOTE;   // 「馬」＝「角」＋成り
 23   public static final int RY=HI+PROMOTE;   // 「竜」＝「飛車」＋成り
```

```
24
25  public static final int SFU=SENTE+FU;     // 「先手の歩」＝「歩」＋「先手」
26  public static final int SKY=SENTE+KY;     // 「先手の香」
27  public static final int SKE=SENTE+KE;     // 「先手の桂」
28  public static final int SGI=SENTE+GI;     // 「先手の銀」
29  public static final int SKI=SENTE+KI;     // 「先手の金」
30  public static final int SKA=SENTE+KA;     // 「先手の角」
31  public static final int SHI=SENTE+HI;     // 「先手の飛」
32  public static final int SOU=SENTE+OU;     // 「先手の玉」
33  public static final int STO=SENTE+TO;     // 「先手のと金」
34  public static final int SNY=SENTE+NY;     // 「先手の成香」
35  public static final int SNK=SENTE+NK;     // 「先手の成桂」
36  public static final int SNG=SENTE+NG;     // 「先手の成銀」
37  public static final int SUM=SENTE+UM;     // 「先手の馬」
38  public static final int SRY=SENTE+RY;     // 「先手の竜」
39
40  public static final int GFU=GOTE +FU;     // 「後手の歩」＝「歩」＋「後手」
41  public static final int GKY=GOTE +KY;     // 「後手の香」
42  public static final int GKE=GOTE +KE;     // 「後手の桂」
43  public static final int GGI=GOTE +GI;     // 「後手の銀」
44  public static final int GKI=GOTE +KI;     // 「後手の金」
45  public static final int GKA=GOTE +KA;     // 「後手の角」
46  public static final int GHI=GOTE +HI;     // 「後手の飛」
47  public static final int GOU=GOTE +OU;     // 「後手の玉」
48  public static final int GTO=GOTE +TO;     // 「後手のと金」
49  public static final int GNY=GOTE +NY;     // 「後手の成香」
50  public static final int GNK=GOTE +NK;     // 「後手の成桂」
51  public static final int GNG=GOTE +NG;     // 「後手の成銀」
52  public static final int GUM=GOTE +UM;     // 「後手の馬」
53  public static final int GRY=GOTE +RY;     // 「後手の竜」
54
55  public static final int WALL=64;          // 盤の外を表すための定数
56
57  // 先手の駒かどうかの判定
58  static public boolean isSente(int koma) {
59    return (koma & SENTE)!=0;
60  }
61
62  // 後手の駒かどうかの判定
63  static public boolean isGote(int koma) {
64    return (koma & GOTE)!=0;
65  }
66
67  // 手番から見て自分の駒かどうか判定
68  static public boolean isSelf(int teban,int koma) {
69    if (teban==SENTE) {
70      return isSente(koma);
71    } else {
```

```
72        return isGote(koma);
73      }
74    }
75
76    // 手番から見て相手の駒かどうか判定
77    static public boolean isEnemy(int teban,int koma) {
78      if (teban==SENTE) {
79        return isGote(koma);
80      } else {
81        return isSente(koma);
82      }
83    }
84
85    // 駒の種類の取得
86    static public int getKomashu(int koma) {
87      // 先手後手のフラグをビット演算でなくせば良い。
88      return koma & 0x0f;
89    }
90
91    // 駒の文字列化用の文字列
92    static public final String komaString[]={
93      "　",
94      "歩",
95      "香",
96      "桂",
97      "銀",
98      "金",
99      "角",
100     "飛",
101     "王",
102     "と",
103     "杏",
104     "圭",
105     "全",
106     "",
107     "馬",
108     "竜"
109   };
110
111   // 駒の文字列化…盤面の表示用
112   static public String toBanString(int koma) {
113     if ( koma==EMPTY ) {
114       return "　 ";
115     } else if ( (koma & SENTE) !=0 ) {
116       // 先手の駒には、" "を頭に追加
117       return " "+komaString[getKomashu(koma)];
118     } else {
119       // 後手の駒には、"v"を頭に追加
```

```
120       return "v"+komaString[getKomashu(koma)];
121     }
122   }
123
124   // 駒の文字列化…持ち駒、手などの表示用
125   static public String toString(int koma) {
126     return komaString[getKomashu(koma)];
127   }
128
129   // 駒が成れるかどうかを表す
130   public static final boolean canPromote[]={
131      false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
132      false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
133      false, true, true, true, true,false, true, true,// 空、先手の歩香桂銀金角飛
134      false,false,false,false,false,false,false,false,// 先手の王、と杏圭全　馬竜
135      false, true, true, true, true,false, true, true,// 空、後手の歩香桂銀金角飛
136      false,false,false,false,false,false,false,false // 後手の王、と杏圭全　馬竜
137   };
138
139   static public boolean canPromote(int koma) {
140     return canPromote[koma];
141   }
142 }
```

＜KomaMoves.java＞

```
  1 package jp.usapyonsoft.lesserpyon;
  2
  3 public interface KomaMoves {
  4   // 通常の８方向の定義(盤面上の動き)
  5   //
  6   //  5   6  7
  7   //      ↑
  8   //  3←駒→4
  9   //      ↓
 10   //  2   1  0
 11   //
 12   // 桂馬飛びの方向の定義(盤面上の動き)
 13   //
 14   //   8    9
 15   //
 16   //      桂
 17   //
 18   //   11  10
 19
 20   // 方向の定義に沿った、「段」の移動の定義
 21   public static final int diffDan[]={
 22      1, 1, 1, 0, 0,-1,-1,-1,-2,-2, 2, 2
 23   };
```

```
24
25  // 方向の定義に沿った、「筋」の移動の定義
26  public static final int diffSuji[]={
27    -1, 0, 1, 1,-1, 1, 0,-1, 1,-1,-1, 1
28  };
29
30  // ある方向にある駒が動けるかどうかを表すテーブル。
31  // 添え字の１つめが方向で、２つめが駒の種類である。
32  // 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、
33  // このテーブルでは false としておく。
34  public static final boolean canMove[][]={
35    // 方向 0 斜め左下への動き
36    {
37     false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
38     false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
39     false,false,false,false, true,false,false,false,// 空、先手の歩香桂銀金角飛
40      true,false,false,false,false,false,false, true,// 先手の王、と杏圭全　馬竜
41     false,false,false,false, true, true,false,false,// 空、後手の歩香桂銀金角飛
42      true, true, true, true, true, true,false, true // 後手の王、と杏圭全　馬竜
43    },
44    // 方向 1 真下への動き
45    {
46     false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
47     false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
48     false,false,false,false,false, true,false,false,// 空、先手の歩香桂銀金角飛
49      true, true, true, true, true,false, true,false,// 先手の王、と杏圭全　馬竜
50     false, true,false,false, true, true,false,false,// 空、後手の歩香桂銀金角飛
51      true, true, true, true, true,false, true,false // 後手の王、と杏圭全　馬竜
52    },
53    // 方向 2 斜め右下への動き
54    {
55     false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
56     false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
57     false,false,false,false, true,false,false,false,// 空、先手の歩香桂銀金角飛
58      true,false,false,false,false,false,false, true,// 先手の王、と杏圭全　馬竜
59     false,false,false,false, true, true,false,false,// 空、後手の歩香桂銀金角飛
60      true, true, true, true, true, true,false, true // 後手の王、と杏圭全　馬竜
61    },
62    // 方向 3 左への動き
63    {
64     false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
65     false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
66     false,false,false,false,false, true,false,false,// 空、先手の歩香桂銀金角飛
67      true, true, true, true, true,false, true,false,// 先手の王、と杏圭全　馬竜
68     false,false,false,false,false, true,false,false,// 空、後手の歩香桂銀金角飛
69      true, true, true, true, true,false, true,false // 後手の王、と杏圭全　馬竜
70    },
71    // 方向 4 右への動き
```

```
72      {
73       false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
74       false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
75       false,false,false,false,false, true,false,false,// 空、先手の歩香桂銀金角飛
76        true, true, true, true, true,false, true,false,// 先手の王、と杏圭全　馬竜
77       false,false,false,false,false, true,false,false,// 空、後手の歩香桂銀金角飛
78        true, true, true, true, true,false, true,false // 後手の王、と杏圭全　馬竜
79      },
80      // 方向 5 斜め左上への動き
81      {
82       false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
83       false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
84       false,false,false,false, true, true,false,false,// 空、先手の歩香桂銀金角飛
85        true, true, true, true, true,false,false, true,// 先手の王、と杏圭全　馬竜
86       false,false,false,false, true,false,false,false,// 空、後手の歩香桂銀金角飛
87        true,false,false,false,false,false,false, true // 後手の王、と杏圭全　馬竜
88      },
89      // 方向 6 真上への動き
90      {
91       false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
92       false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
93       false, true,false,false, true, true,false,false,// 空、先手の歩香桂銀金角飛
94        true, true, true, true, true,false, true,false,// 先手の王、と杏圭全　馬竜
95       false,false,false,false,false, true,false,false,// 空、後手の歩香桂銀金角飛
96        true, true, true, true, true,false, true,false // 後手の王、と杏圭全　馬竜
97       },
98       // 方向 7 斜め右上への動き
99       {
100       false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
101       false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
102       false,false,false,false, true, true,false,false,// 空、先手の歩香桂銀金角飛
103        true, true, true, true, true,false,false, true,// 先手の王、と杏圭全　馬竜
104       false,false,false,false, true,false,false,false,// 空、後手の歩香桂銀金角飛
105        true,false,false,false,false,false,false, true // 後手の王、と杏圭全　馬竜
106      },
107      // 方向 8 先手の桂馬飛び
108      {
109       false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
110       false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
111       false,false,false, true,false,false,false,false,// 空、先手の歩香桂銀金角飛
112       false,false,false,false,false,false,false,false,// 先手の王、と杏圭全　馬竜
113       false,false,false,false,false,false,false,false,// 空、後手の歩香桂銀金角飛
114       false,false,false,false,false,false,false,false // 後手の王、と杏圭全　馬竜
115      },
116      // 方向 9 先手の桂馬飛び
117      {
118       false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
119       false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
```

```
120       false,false,false, true,false,false,false,false,// 空、先手の歩香桂銀金角飛
121       false,false,false,false,false,false,false,false,// 先手の王、と杏圭全　馬竜
122       false,false,false,false,false,false,false,false,// 空、後手の歩香桂銀金角飛
123       false,false,false,false,false,false,false,false // 後手の王、と杏圭全　馬竜
124     },
125     // 方向 10 後手の桂馬飛び
126     {
127       false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
128       false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
129       false,false,false,false,false,false,false,false,// 空、先手の歩香桂銀金角飛
130       false,false,false,false,false,false,false,false,// 先手の王、と杏圭全　馬竜
131       false,false,false, true,false,false,false,false,// 空、後手の歩香桂銀金角飛
132       false,false,false,false,false,false,false,false // 後手の王、と杏圭全　馬竜
133     },
134     // 方向 11 後手の桂馬飛び
135     {
136       false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
137       false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
138       false,false,false,false,false,false,false,false,// 空、先手の歩香桂銀金角飛
139       false,false,false,false,false,false,false,false,// 先手の王、と杏圭全　馬竜
140       false,false,false, true,false,false,false,false,// 空、後手の歩香桂銀金角飛
141       false,false,false,false,false,false,false,false // 後手の王、と杏圭全　馬竜
142     }
143   };
144
145   // ある方向にある駒が飛べるかどうかを表すテーブル。
146   // 添え字の１つめが方向で、２つめが駒の種類である。
147   // 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。
148   static final public boolean canJump[][]={
149     // 方向 0 斜め左下への動き
150     {
151       false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
152       false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
153       false,false,false,false,false,false, true,false,// 空、先手の歩香桂銀金角飛
154       false,false,false,false,false,false, true,false,// 先手の王、と杏圭全　馬竜
155       false,false,false,false,false,false, true,false,// 空、後手の歩香桂銀金角飛
156       false,false,false,false,false,false, true,false // 後手の王、と杏圭全　馬竜
157     },
158     // 方向 1 真下への動き
159     {
160       false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
161       false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
162       false,false,false,false,false,false,false, true,// 空、先手の歩香桂銀金角飛
163       false,false,false,false,false,false,false, true,// 先手の王、と杏圭全　馬竜
164       false,false, true,false,false,false,false, true,// 空、後手の歩香桂銀金角飛
165       false,false,false,false,false,false,false, true // 後手の王、と杏圭全　馬竜
166     },
167     // 方向 2 斜め右下への動き
```

```
168     {
169      false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
170      false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
171      false,false,false,false,false,false, true,false,// 空、先手の歩香桂銀金角飛
172      false,false,false,false,false,false, true,false,// 先手の王、と杏圭全　馬竜
173      false,false,false,false,false,false, true,false,// 空、後手の歩香桂銀金角飛
174      false,false,false,false,false,false, true,false // 後手の王、と杏圭全　馬竜
175     },
176     // 方向 3 左への動き
177     {
178      false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
179      false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
180      false,false,false,false,false,false,false, true,// 空、先手の歩香桂銀金角飛
181      false,false,false,false,false,false,false, true,// 先手の王、と杏圭全　馬竜
182      false,false,false,false,false,false,false, true,// 空、後手の歩香桂銀金角飛
183      false,false,false,false,false,false,false, true // 後手の王、と杏圭全　馬竜
184     },
185     // 方向 4 右への動き
186     {
187      false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
188      false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
189      false,false,false,false,false,false,false, true,// 空、先手の歩香桂銀金角飛
190      false,false,false,false,false,false,false, true,// 先手の王、と杏圭全　馬竜
191      false,false,false,false,false,false,false, true,// 空、後手の歩香桂銀金角飛
192      false,false,false,false,false,false,false, true // 後手の王、と杏圭全　馬竜
193     },
194     // 方向 5 斜め左上への動き
195     {
196      false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
197      false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
198      false,false,false,false,false,false, true,false,// 空、先手の歩香桂銀金角飛
199      false,false,false,false,false,false, true,false,// 先手の王、と杏圭全　馬竜
200      false,false,false,false,false,false, true,false,// 空、後手の歩香桂銀金角飛
201      false,false,false,false,false,false, true,false // 後手の王、と杏圭全　馬竜
202     },
203     // 方向 6 真上への動き
204     {
205      false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
206      false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
207      false,false, true,false,false,false,false, true,// 空、先手の歩香桂銀金角飛
208      false,false,false,false,false,false,false, true,// 先手の王、と杏圭全　馬竜
209      false,false,false,false,false,false,false, true,// 空、後手の歩香桂銀金角飛
210      false,false,false,false,false,false,false, true // 後手の王、と杏圭全　馬竜
211     },
212     // 方向 7 斜め右上への動き
213     {
214      false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
215      false,false,false,false,false,false,false,false,// 先手でも後手でもない駒
```

```
216      false,false,false,false,false,false, true,false,// 空、先手の歩香桂銀金角飛
217      false,false,false,false,false,false, true,false,// 先手の王、と杏圭全　馬竜
218      false,false,false,false,false,false, true,false,// 空、後手の歩香桂銀金角飛
219      false,false,false,false,false,false, true,false // 後手の王、と杏圭全　馬竜
220    }
221    // 桂馬の方向に飛ぶ駒はないので、以下は省略。
222  };
223 }
```

＜Kyokumen.java＞

```
  1 package jp.usapyonsoft.lesserpyon;
  2
  3 import java.util.Random;
  4 import java.util.Vector;
  5
  6 class Kyokumen implements Constants, Cloneable {
  7
  8  // 盤面
  9  int ban[][];
 10
 11  // 持ち駒
 12  Vector hand[];
 13
 14  // 手番
 15  int teban = SENTE;
 16
 17  static int e;
 18  static int sum = 0;
 19  static int sum2 = 0;
 20  static Random random = new Random();
 21  public int randomValue = random.nextInt(1000) - 500;
 22
 23  public Kyokumen() {
 24          ban = new int[7][7];
 25          hand = new Vector[2];
 26          hand[0] = new Vector();
 27          hand[1] = new Vector();
 28          e = randomValue;
 29  }
 30
 31  // 局面のコピーを行う
 32  public Object clone() {
 33          Kyokumen k = new Kyokumen();
 34
 35          // 盤面のコピー
 36          for (int suji = 0; suji < 7; suji++) {
 37                  for (int dan = 0; dan < 7; dan++) {
 38                          k.ban[suji][dan] = ban[suji][dan];
```

```
39                  }
40          }
41
42          // 持ち駒のコピー
43          k.hand[0] = (Vector) hand[0].clone();
44          k.hand[1] = (Vector) hand[1].clone();
45
46          // 手番のコピー
47          k.teban = teban;
48
49          return k;
50  }
51
52  // 局面が同一かどうか
53  public boolean equals(Object o) {
54          Kyokumen k = (Kyokumen) o;
55          if (k == null)
56                  return false;
57          return equals(k);
58  }
59
60  // 局面が同一かどうか
61  public boolean equals(Kyokumen k) {
62          // 手番の比較
63          if (teban != k.teban) {
64                  return false;
65          }
66
67          // 盤面の比較
68          // 各マスについて…
69          for (int suji = 1; suji <= 5; suji++) {
70                  for (int dan = 1; dan <= 5; dan++) {
71                  // 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
72                          // 同じかどうか比較する。
73                          if (!(ban[suji][dan] == k.ban[suji][dan])) {
74                                  // 違っていたら、false を返す。
75                                  return false;
76                          }
77                  }
78          }
79
80          // 持ち駒の比較
81          // 駒の種類ごとに枚数が同じかどうかを比較してみる。
82          // 比較用の配列を準備。
83          int handSente[] = new int[Koma.HI + 1];
84          int handGote[] = new int[Koma.HI + 1];
85          int compareHandSente[] = new int[Koma.HI + 1];
86          int compareHandGote[] = new int[Koma.HI + 1];
```

```
87
88          // 各配列は 0 で初期化されている。
89          // hand に現れた駒を、駒の種類毎に数えていく。
90
91          // まず、自分の先手の持ち駒
92          for (int i = 0; i < hand[0].size(); i++) {
93                  Integer koma = (Integer) hand[0].elementAt(i);
94                  int komaShu = Koma.getKomashu(koma.intValue());
95                  handSente[komaShu]++;
96          }
97          // 自分の後手の持ち駒
98          for (int i = 0; i < hand[1].size(); i++) {
99                  Integer koma = (Integer) hand[1].elementAt(i);
100                 int komaShu = Koma.getKomashu(koma.intValue());
101                 handGote[komaShu]++;
102         }
103
104         // 比較対象の先手の持ち駒
105         for (int i = 0; i < k.hand[0].size(); i++) {
106                 Integer koma = (Integer) k.hand[0].elementAt(i);
107                 int komaShu = Koma.getKomashu(koma.intValue());
108                 compareHandSente[komaShu]++;
109         }
110         // 比較対象の後手の持ち駒
111         for (int i = 0; i < k.hand[1].size(); i++) {
112                 Integer koma = (Integer) k.hand[1].elementAt(i);
113                 int komaShu = Koma.getKomashu(koma.intValue());
114                 compareHandGote[komaShu]++;
115         }
116
117         // 持ち駒の枚数を比較する。
118         for (int i = Koma.FU; i <= Koma.HI; i++) {
119                 if (handSente[i] != compareHandSente[i])
120                         return false;
121                 if (handGote[i] != compareHandGote[i])
122                         return false;
123         }
124
125         // 完全に一致した。
126         return true;
127 }
128
129 // ある位置にある駒を取得する
130 public int get(Position p) {
131         // 盤外なら、「盤外＝壁」を返す
132         if (p.suji < 1 || 5 < p.suji || p.dan < 1 || 5 < p.dan) {
133                 return Koma.WALL;
134         }
```

```
135         return ban[p.suji][p.dan];
136 }
137
138 // ある位置にある駒を置く。
139 public void put(Position p, int koma) {
140         ban[p.suji][p.dan] = koma;
141 }
142
143 // 与えられた手で一手進めてみる。
144 public void move(Te te) {
145         // 駒の行き先に駒があったなら…
146         if (get(te.to) != Koma.EMPTY) {
147                 // 持ち駒にする
148                 if (Koma.isSente(get(te.to))) {
149                         // 取った駒が先手の駒なら後手の持ち駒に。
150                         int koma = get(te.to);
151                         // 成りなどのフラグ、先手・後手の駒のフラグをクリア。
152                         koma = koma & 0x07;
153                         // 後手の駒としてのフラグをセット
154                         koma = koma | GOTE;
155                         hand[1].add(new Integer(koma));
156                 } else {
157                         // 取った駒が後手の駒なら先手の持ち駒に。
158                         int koma = get(te.to);
159                         // 成りなどのフラグ、先手・後手の駒のフラグをクリア。
160                         koma = koma & 0x07;
161                         // 先手の駒としてのフラグをセット
162                         koma = koma | SENTE;
163                         hand[0].add(new Integer(koma));
164                 }
165         }
166         if (te.from.suji == 0) {
167                 // 持ち駒を打った
168                 if (Koma.isSente(te.koma)) {
169                         // 先手の駒なら、先手の持ち駒を減らす。
170                         for (int i = 0; i < hand[0].size(); i++) {
171                         int koma = ((Integer) hand[0].elementAt(i)).intValue();
172                                 if (koma == te.koma) {
173                                         hand[0].removeElementAt(i);
174                                         break;
175                                 }
176                         }
177                 } else {
178                         // 後手の駒を打ったはずなので、後手の持ち駒を減らす
179                         for (int i = 0; i < hand[1].size(); i++) {
180                         int koma = ((Integer) hand[1].elementAt(i)).intValue();
181                                 if (koma == te.koma) {
182                                         hand[1].removeElementAt(i);
```

```
183                                             break;
184                                     }
185                             }
186                     }
187             } else {
188                     // 盤上の駒を進めた→元の位置は、EMPTY に。
189                     put(te.from, Koma.EMPTY);
190             }
191             // 駒を移動先に進める。
192             int koma = te.koma;
193             if (te.promote) {
194                     // 「成り」の処理
195                     koma = koma | Koma.PROMOTE;
196             }
197             put(te.to, koma);
198     }
199
200     // 玉を探して、位置を返す
201     public Position searchGyoku(int teban) {
202             // 探す駒は、teban 側の玉
203             int toSearch = teban | Koma.OU;
204             // 筋、段でループ
205             for (int suji = 1; suji <= 5; suji++) {
206                     for (int dan = 1; dan <= 5; dan++) {
207                             if (ban[suji][dan] == toSearch) {
208                                     // 見つかった位置を返す。
209                                     return new Position(suji, dan);
210                             }
211                     }
212             }
213             // 見つからなかった…。
214             // 駒の利きの届かない盤外を返す。
215             // 0,0 などに設定すると、1 一にいる相手駒の利きを見つけてしまう可能性がある。
216             return new Position(-2, -2);
217     }
218
219     // 局面を評価するための、駒の価値。
220     // 先手の駒はプラス点、後手の駒はマイナス点にする。
221
222 // static final int komaValue_sente[]={
223 //         0,     0,     0,     0,     0,     0,     0,     0, // 何もない場所及び
224 //         0,     0,     0,     0,     0,     0,     0,     0, // 先手でも後手でもない駒
225 //         0, sum = (100 + e),  600,  700, 1000, 1200, 1800, 2000,
                                                                    // 何もない場所、先手の歩～飛車
226 //   10000, sum2 = (1200 + e), 1200, 1200, 1200,     0, 2000, 2200,
                                                                        // 先手玉、及びと～竜
227 //         0, -sum, -600, -700,-1000,-1200,-1800,-2000, // 何もない場所、後手の歩～飛車
228 //  -10000,-sum2,-1200,-1200,-1200,     0,-2000,-2200  // 後手玉、及びと～竜
```

```
229 // };
230
231   static final int komaValue_sente[]={
232         0,     0,     0,     0,     0,     0,     0,     0, // 何もない場所及び
233         0,     0,     0,     0,     0,     0,     0,     0, // 先手でも後手でもない駒
234         0,   100,   600,   700,  1000,  1200,  1000,  1500, // 何もない場所、後手の歩～飛車
235     10000,  1200,  1200,  1200,  1200,     0,  1000,  1500, // 先手玉、及びと～竜
236         0,  -100,  -600,  -700, -1000, -1200, -1000, -1500, // 何もない場所、先手の歩～飛車
237    -10000, -1200, -1200, -1200, -1200,     0, -1000, -1500  // 後手玉、及びと～竜
238   };
239
240   static final int komaValue_gote[]={
241         0,     0,     0,     0,     0,     0,     0,     0, // 何もない場所及び
242         0,     0,     0,     0,     0,     0,     0,     0, // 先手でも後手でもない駒
243         0,   100,   600,   700,  1000,  1200,  1800,  2000, // 何もない場所、先手の歩～飛車
244     10000,  1200,  1200,  1200,  1200,     0,  2000,  2200, // 先手玉、及びと～竜
245         0,  -100,  -600,  -700, -1000, -1200, -1800, -2000, // 何もない場所、後手の歩～飛車
246    -10000, -1200, -1200, -1200, -1200,     0, -2000, -2200  // 後手玉、及びと～竜
247   };
248
249 // static final int komaValue[]={
250 //       0,     0,     0,     0,     0,     0,     0,     0, // 何もない場所及び
251 //       0,     0,     0,     0,     0,     0,     0,     0, // 先手でも後手でもない駒
252 //       0,   100,   600,   700,  1000,  1200,  1800,  2000, // 何もない場所、先手の歩～飛車
253 //   10000,  1200,  1200,  1200,  1200,     0,  2000,  2200, // 先手玉、及びと～竜
254 //       0,  -100,  -600,  -700, -1000, -1200, -1800, -2000, // 何もない場所、後手の歩～飛車
255 //  -10000, -1200, -1200, -1200, -1200,     0, -2000, -2200  // 後手玉、及びと～竜
256 // };
257
258   // 局面を評価する関数。
259   public int evaluate() {
260           // 局面を評価した値…0でまず初期化。
261           int eval = 0;
262           // 盤面上の駒の価値を全部加算
263
264           if(teban == SENTE) {
265                   for (int suji = 1; suji <= 5; suji++) {
266                           for (int dan = 1; dan <= 5; dan++) {
267                                   int koma = ban[suji][dan];
268                                   eval = eval + komaValue_sente[koma];
269                           }
270                   }
271           }
272           if(teban == GOTE) {
273                   for (int suji = 1; suji <= 5; suji++) {
274                           for (int dan = 1; dan <= 5; dan++) {
275                                   int koma = ban[suji][dan];
276                                   eval = eval + komaValue_gote[koma];
```

```
277                             }
278                     }
279             } else {
280
281             }
282
283             // 先手、後手の持ち駒の価値を全部加算
284             for (int i = 0; i < 2; i++) {
285                     for (int j = 0; j < hand[i].size(); j++) {
286                             int koma = ((Integer) hand[i].elementAt(j)).intValue();
287                             eval = eval + komaValue_sente[koma] + komaValue_gote[koma];
288                     }
289             }
290
291             eval += random.nextInt(1000) - 500;
292
293             // これで、盤上の駒と持ち駒の価値を加算し終わった。
294             return eval;
295     }
296
297     // CSA 形式の棋譜ファイル文字列
298     static final String csaKomaTbl[] = {
299             "   ", "FU", "KY", "KE", "GI", "KI", "KA", "HI",
300             "OU", "TO", "NY", "NK ", "NG", "", "UM", "RY",
301             "", "+FU", "+KY", "+KE", "+GI", "+KI", "+KA", "+HI",
302             "+OU", "+TO", "+NY", "+NK", "+NG", "", "+UM", "+RY",
303             "", "-FU", "-KY", "-KE", "-GI", "-KI", "-KA", "-HI",
304             "-OU", "-TO", "-NY", "-NK", "-NG", "", "-UM", "-RY"
305     };
306
307     // CSA 形式の棋譜ファイルから、局面を読み込む
308     public void ReadCsaKifu(String[] csaKifu) {
309     // 持ち駒…枚数の配列にしておく。
310     int motigoma[][] = new int[2][Koma.HI + 1];
311
312     // 駒箱に入っている残りの駒。残りを全て持ち駒にする際などに使用する。
313     int restKoma[] = new int[Koma.HI + 1];
314
315     // 持ち駒を空に。
316     for (int i = 0; i <= Koma.HI; i++) {
317             motigoma[0][i] = 0;
318             motigoma[1][i] = 0;
319     }
320
321     // 駒箱に入っている駒＝その種類の駒の枚数
322     //    restKoma[Koma.FU]=18;
323     //    restKoma[Koma.KY]=4;
324     //    restKoma[Koma.KE]=4;
```

```
325  //    restKoma[Koma.GI]=4;
326  //    restKoma[Koma.KI]=4;
327  //    restKoma[Koma.KA]=2;
328  //    restKoma[Koma.HI]=2;
329  restKoma[Koma.FU] = 2;
330  restKoma[Koma.GI] = 2;
331  restKoma[Koma.KI] = 2;
332  restKoma[Koma.KA] = 2;
333  restKoma[Koma.HI] = 2;
334
335  // 盤面を空に初期化
336  //    for(int suji=1;suji<=9;suji++) {
337  //      for(int dan=1;dan<=9;dan++) {
338  //        ban[suji][dan]=Koma.EMPTY;
339  //      }
340  //    }
341  for (int suji = 1; suji <= 5; suji++) {
342          for (int dan = 1; dan <= 5; dan++) {
343                  ban[suji][dan] = Koma.EMPTY;
344          }
345  }
346
347  // 文字列から読み込み
348  for (int i = 0; i < csaKifu.length; i++) {
349  String line = csaKifu[i];
350  System.out.println("" + i + "  " + line);
351  if (line.startsWith("P+")) {
352          if (line.equals("P+00AL")) {
353                  // 残りの駒は全部先手の持ち駒
354                  for (int koma = Koma.FU; koma <= Koma.HI; koma++) {
355                          motigoma[0][koma] = restKoma[koma];
356                  }
357          } else {
358                  // 先手の持ち駒
359                  for (int j = 0; j <= line.length() - 6; j += 4) {
360                          int koma = 0;
361                          String komaStr = line.substring(j + 2 + 2, j + 2 + 4);
362                          for (int k = Koma.FU; k <= Koma.HI; k++) {
363                                  if (komaStr.equals(csaKomaTbl[k])) {
364                                          koma = k;
365                                          break;
366                                  }
367                          }
368                          motigoma[0][koma]++;
369                  }
370          }
371  } else if (line.startsWith("P-")) {
372          if (line.equals("P-00AL")) {
```

```
373                     // 残りの駒は全部後手の持ち駒
374                     for (int koma = Koma.FU; koma <= Koma.HI; koma++) {
375                             motigoma[1][koma] = restKoma[koma];
376                     }
377             } else {
378             // 後手の持ち駒
379             for (int j = 0; j < line.length(); j += 4) {
380                     int koma = 0;
381                     for (int k = Koma.FU; k <= Koma.HI; k++) {
382                             if (line.substring(j + 2, j + 4).equals(csaKomaTbl[k])) {
383                                     koma = k;
384                                     break;
385                             }
386                     }
387                     motigoma[1][koma]++;
388             }
389     }
390     } else if (line.startsWith("P")) {
391             // 盤面の表現
392             // P1～P9 まで。
393             String danStr = line.substring(1, 2);
394             int dan = 0;
395             try {
396                     dan = Integer.parseInt(danStr);
397             } catch (Exception e) {
398                     // …握りつぶすことにしておく。
399             }
400             String komaStr;
401             for (int suji = 1; suji <= 5; suji++) {
402                     // ややこしいが、左側が９筋、右側が１筋…
403                     // 文字列の頭の方が９筋で、後ろの方が１筋。
404                     // そのため、読み込みの時に逆さに読み込む。
405                     komaStr = line.substring(2 + (5 - suji) * 3, 2 + (5 - suji) * 3 + 3);
406                     int koma = Koma.EMPTY;
407                     for (int k = Koma.EMPTY; k <= Koma.GRY; k++) {
408                             if (komaStr.equals(csaKomaTbl[k])) {
409                             koma = k;
410                             // 成のフラグを取って、残りの駒から
411                             // その種類の駒を一枚ひいておく。
412                             restKoma[(Koma.getKomashu(koma) & ~Koma.PROMOTE)]--;
413                             break;
414                             }
415                     }
416                             ban[suji][dan] = koma;
417                     }
418             } else if (line.equals("-")) {
419                     teban = GOTE;
420             } else if (line.equals("+")) {
```

```
421                     teban = SENTE;
422             }
423     }
424     // 持ち駒を hand にしまう。
425     for (int i = Koma.FU; i < Koma.HI; i++) {
426             for (int j = 0; j < motigoma[0][i]; j++) {
427                     hand[0].add(new Integer(i | SENTE));
428             }
429             for (int j = 0; j < motigoma[1][i]; j++) {
430                     hand[1].add(new Integer(i | GOTE));
431             }
432     }
433     }
434
435     // 局面を表示用に文字列化
436     public String toString() {
437             String s = "";
438             // 後手持ち駒表示
439             s += "後手持ち駒 ";
440             for (int i = 0; i < hand[1].size(); i++) {
441                     s += Koma.toString(((Integer) hand[1].elementAt(i)).intValue());
442             }
443             s += "¥n";
444             // 盤面表示
445             s += "  ５   ４   ３   ２   １¥n";
446             s += "+---+---+---+---+---+¥n";
447             for (int dan = 1; dan <= 5; dan++) {
448                     for (int suji = 5; suji >= 1; suji--) {
449                             s += "|";
450                             s += Koma.toBanString(ban[suji][dan]);
451                     }
452                     s += "|";
453                     s += danStr[dan];
454                     s += "¥n";
455                     s += "+---+---+---+---+---+¥n";
456             }
457             // 先手持ち駒表示
458             s += "先手持ち駒 ";
459             for (int i = 0; i < hand[0].size(); i++) {
460                     s += Koma.toString(((Integer) hand[0].elementAt(i)).intValue());
461             }
462             s += "¥n";
463
464             return s;
465     }
466
467     public int randomValue() {
468             return e;
```

```
469   }
470 }
```

＜Main.java＞

```
  1 package jp.usapyonsoft.lesserpyon;
  2 import java.io.BufferedReader;
  3 import java.io.File;
  4 import java.io.FileReader;
  5 import java.util.Vector;
  6
  7 public class Main implements Constants {
  8   // 初期盤面を与える
  9   static final int ShokiBanmen[][]={
 10 //{Koma.GKY,Koma.GKE,Koma.GGI,Koma.GKI,Koma.GOU,Koma.GKI,Koma.GGI,Koma.GKE,Koma.GKY},
 11 //{Koma.EMP,Koma.GHI,Koma.EMP,Koma.EMP,Koma.EMP,Koma.EMP,Koma.EMP,Koma.GKA,Koma.EMP},
 12 //{Koma.GFU,Koma.GFU,Koma.GFU,Koma.GFU,Koma.GFU,Koma.GFU,Koma.GFU,Koma.GFU,Koma.GFU},
 13 //{Koma.EMP,Koma.EMP,Koma.EMP,Koma.EMP,Koma.EMP,Koma.EMP,Koma.EMP,Koma.EMP,Koma.EMP},
 14 //{Koma.EMP,Koma.EMP,Koma.EMP,Koma.EMP,Koma.EMP,Koma.EMP,Koma.EMP,Koma.EMP,Koma.EMP},
 15 //{Koma.EMP,Koma.EMP,Koma.EMP,Koma.EMP,Koma.EMP,Koma.EMP,Koma.EMP,Koma.EMP,Koma.EMP},
 16 //{Koma.SFU,Koma.SFU,Koma.SFU,Koma.SFU,Koma.SFU,Koma.SFU,Koma.SFU,Koma.SFU,Koma.SFU},
 17 //{Koma.EMP,Koma.SKA,Koma.EMP,Koma.EMP,Koma.EMP,Koma.EMP,Koma.EMP,Koma.SHI,Koma.EMP},
 18 //{Koma.SKY,Koma.SKE,Koma.SGI,Koma.SKI,Koma.SOU,Koma.SKI,Koma.SGI,Koma.SKE,Koma.SKY},
 19       {Koma.GHI,Koma.GKA,Koma.GGI,Koma.GKI,Koma.GOU},
 20       {Koma.EMP,Koma.EMP,Koma.EMP,Koma.EMP,Koma.GFU},
 21       {Koma.EMP,Koma.EMP,Koma.EMP,Koma.EMP,Koma.EMP},
 22       {Koma.SFU,Koma.EMP,Koma.EMP,Koma.EMP,Koma.EMP},
 23       {Koma.SOU,Koma.SKI,Koma.SGI,Koma.SKA,Koma.SHI},
 24   };
 25
 26   // player[0]が先手が誰か、player[1]が後手が誰か。
 27   static Player player[]=new Player[2];
 28
 29   static Vector kyokumenRireki=new Vector();
 30
 31   // 使い方を表示する。
 32   static void usage() {
 33     System.out.println("使い方 ");
 34     System.out.println("例 先手　人間　後手　コンピュータの場合");
 35     System.out.println("java jp.usapyonsoft.lesserpyon.Main HUMAN CPU");
 36     System.out.println("");
 37     System.out.println("初期局面を与えて対局開始することも可能です。");
 38     System.out.println("例 kyokumen.csa に初期局面が入っているとした場合");
 39     System.out.println("java jp.usapyonsoft.lesserpyon.Main HUMAN CPU kyokumen.csa");
 40   }
 41
 42   // 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。
 43   // 誰か、は、人間ならば "HUMAN" CPU ならば "CPU"を与える。
 44   // 初期局面が与えられなかった場合、平手の初期局面から。
```

```
45  public static void main(String argv[]) {
46
47    for(int ebi = 0; ebi<100 ; ebi++) {
48            // ループのために格納履歴を初期化
49            Vector kyokumenRireki=new Vector();
50
51    // 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
52    if (argv.length<2) {
53      // 引数が足りないようなら、使い方を表示して終わり。
54      usage();
55      return;
56    }
57
58    // 先手番が誰かを設定。
59    if (argv[0].equals("HUMAN")) {
60      player[0]=new Human();
61    } else if (argv[0].equals("CPU")) {
62      player[0]=new Sikou();
63    } else {
64      // 引数がおかしいようなら、使い方を表示して終わり。
65      usage();
66      return;
67    }
68
69    // 後手番が誰かを設定。
70    if (argv[1].equals("HUMAN")) {
71      player[1]=new Human();
72    } else if (argv[1].equals("CPU")) {
73      player[1]=new Sikou();
74    } else {
75      // 引数がおかしいようなら、使い方を表示して終わり。
76      usage();
77      return;
78    }
79
80    try {
81      Kyokumen k=new Kyokumen();
82      if (argv.length==2) {
83        // 引数の指定がない場合、初期配置を使う。
84        // 先手番
85        k.teban=SENTE;
86        for(int dan=1;dan<=5;dan++) {
87          for(int suji=5;suji>=1;suji--) {
88            k.ban[suji][dan]=ShokiBanmen[dan-1][5-suji];
89          }
90        }
91      } else {
92        // 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。
```

```
93          String csaFileName=argv[2];
94          File f=new File(csaFileName);
95          BufferedReader in=new BufferedReader(new FileReader(f));
96          Vector v=new Vector();
97          String s;
98          while((s=in.readLine())!=null) {
99            System.out.println("Read "+s);
100           v.add(s);
101         }
102         String csaKifu[]=new String[v.size()];
103         v.copyInto(csaKifu);
104         k.ReadCsaKifu(csaKifu);
105       }
106
107       // 対戦のメインループ
108       while(true) {
109         // 現在の局面を、局面の履歴に保存する。
110         kyokumenRireki.add(k.clone());
111         // 現在の局面での合法手を生成
112         Vector v=GenerateMoves.generateLegalMoves(k);
113         if (v.size()==0) {
114           // 手番の側の負け
115           if (k.teban==SENTE) {
116             System.out.println("後手の勝ち！");
117           } else {
118             System.out.println("先手の勝ち！");
119           }
120           // 対局終了
121           break;
122         }
123         // 千日手のチェック…連続王手の千日手には未対応。
124         // 同一局面が何回出てきたか？
125         int sameKyokumen=0;
126         for(int i=0;i<kyokumenRireki.size();i++) {
127           // 同一局面だったら…
128           if (kyokumenRireki.elementAt(i).equals(k)) {
129             // 同一局面の出てきた回数を増やす
130             sameKyokumen++;
131           }
132         }
133         if (sameKyokumen>=4) {
134           // 同一局面４回以上の繰り返しなので、千日手。
135           System.out.println("千日手です。");
136           // 対局終了
137           break;
138         }
139
140         // 局面を表示。
```

```
141 //        System.out.println(k.toString());
142         // ついでに、局面の評価値を表示
143 //        System.out.println("現在の局面の評価値 "+k.evaluate());
144         // 次の手を手番側のプレイヤーから取得
145         Te te;
146         if (k.teban==SENTE) {
147           te=player[0].getNextTe(k);
148         } else {
149           te=player[1].getNextTe(k);
150         }
151         // 指された手を表示
152 //        System.out.println(te.toString());
153
154         // 合法手でない手を指した場合、即負け
155         if (!v.contains(te)) {
156           System.out.println("合法手でない手が指されました。");
157           // 手番の側の負け
158           if (k.teban==SENTE) {
159             System.out.println("後手の勝ち！");
160           } else {
161             System.out.println("先手の勝ち！");
162           }
163           // 対局終了
164           break;
165         }
166         // 指された手で局面を進める。
167         k.move(te);
168         // move では、手番が変わらないので、局面の手番を変更する。
169         if (k.teban==SENTE) {
170           k.teban=GOTE;
171         } else {
172           k.teban=SENTE;
173         }
174       }
175       // 対局終了。最後の局面を表示して、終わる。
176 //      System.out.println("対局終了です。");
177 //      System.out.println("最後の局面は…");
178 //      System.out.println(k.toString());
179
180       /**
181        * 乱数値
182        */
183 //      System.out.println("乱数"+ k.randomValue() + "で実行しました. ");
184
185       /*
186        * 詰み判定考察のために作成
187        */
188 //      System.out.println("現在の局面の評価値 "+k.evaluate());
```

```
189
190     } catch(Exception ex) {
191       ex.printStackTrace();
192     }
193   }
194
195   }
196
197 }
```

＜Player.java＞

```
 1 package jp.usapyonsoft.lesserpyon;
 2
 3 public interface Player {
 4   Te getNextTe(Kyokumen k);
 5 }
```

＜Position.java＞

```
 1 package jp.usapyonsoft.lesserpyon;
 2
 3 // 駒の位置を表すクラス
 4 class Position implements Cloneable,KomaMoves {
 5   // 筋
 6   public int suji;
 7   // 段
 8   public int dan;
 9
10   // コンストラクタ
11   public Position() {
12     suji=0;
13     dan=0;
14   }
15
16   // コンストラクタ
17   public Position(int _suji,int _dan) {
18     suji=_suji;
19     dan=_dan;
20   }
21
22   // 同一性比較用メソッド
23   public boolean equals(Position p) {
24     return (p.suji==suji && p.dan==dan);
25   }
26   public boolean equals(Object o) {
27     Position p=(Position)o;
28     if (p==null) return false;
29     return equals(p);
30   }
```

```
31
32   // コピーを返す
33   public Object clone() {
34     return new Position(suji,dan);
35   }
36
37   // ある方向への動きを行う
38   public void add(int diffSuji,int diffDan) {
39     suji+=diffSuji;
40     dan+=diffDan;
41   }
42
43   // ある方向への逆向きの動きを行う
44   public void sub(int diffSuji,int diffDan) {
45     suji-=diffSuji;
46     dan-=diffDan;
47   }
48
49   // ある方向への動きを行う
50   public void add(int direct) {
51     add(diffSuji[direct],diffDan[direct]);
52   }
53
54   // ある方向への逆向きの動きを行う
55   public void sub(int direct) {
56     sub(diffSuji[direct],diffDan[direct]);
57   }
58 }
```

＜Sikou.java＞

```
 1 package jp.usapyonsoft.lesserpyon;
 2 import java.util.Vector;
 3
 4 // コンピュータの思考ルーチン
 5 public class Sikou implements Player,Constants {
 6   // ∞を表すための定数
 7   static final int INFINITE=99999999;
 8
 9   // 読みの深さ
10   static final int DEPTH_MAX=4;
11
12   // 読みの最大深さ…これ以上の読みは絶対に不可能。
13   static final int LIMIT_DEPTH=16;
14
15   // αβカットを起こす関係で、ランダム着手は出来なくなる。
16   // 詳細は解説にて。
17
18   // 最善手順を格納する配列
```

```
19  Te best[][]=new Te[LIMIT_DEPTH][LIMIT_DEPTH];
20
21  int leaf=0;
22  int node=0;
23
24
25  public Sikou() {
26  }
27
28  int negaMax(Te t,Kyokumen k,int alpha,int beta,int depth,int depthMax) {
29    // 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。
30    if (depth>=depthMax) {
31      leaf++;
32      if (k.teban==SENTE) {
33        return k.evaluate();
34      } else {
35        return -k.evaluate();
36      }
37    }
38    node++;
39    // 現在の局面での合法手を生成
40    Vector v=GenerateMoves.generateLegalMoves(k);
41
42    // 現在の指し手の候補手の評価値を入れる。
43    int value=-INFINITE;
44
45    // 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。
46    for(int i=0;i<v.size();i++) {
47      // 合法手を取り出す。
48      Te te=(Te)v.elementAt(i);
49
50      // その手で一手進めた局面を作ってみる。
51      Kyokumen nextKyokumen=(Kyokumen)k.clone();
52      nextKyokumen.move(te);
53      // move では、先手後手を入れ替えないので…。
54      if (nextKyokumen.teban==SENTE) {
55        nextKyokumen.teban=GOTE;
56      } else {
57        nextKyokumen.teban=SENTE;
58      }
59
60      // その局面の評価値を、さらに先読みして得る。
61      Te tmpTe=new Te(0,new Position(0,0),new Position(0,0),false);
62      int eval=-negaMax(tmpTe,nextKyokumen,-beta,-alpha,depth+1,depthMax);
63
64      // 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？
65      if (eval>value || value == -INFINITE) {
66        // 返す値を更新
```

```
 67          value=eval;
 68          // α値も更新
 69          if (eval>alpha) {
 70            alpha=eval;
 71          }
 72          // 最善手を更新
 73          best[depth][depth]=te;
 74          t.koma   =te.koma;
 75          t.from   =te.from;
 76          t.to     =te.to;
 77          t.promote=te.promote;
 78          // 最善手順を更新
 79          for(int j=depth+1;j<depthMax;j++) {
 80            best[depth][j]=best[depth+1][j];
 81          }
 82          // βカットの条件を満たしていたら、ループ終了。
 83          if (eval>=beta) {
 84            break;
 85          }
 86        }
 87      }
 88      return value;
 89    }
 90
 91    public Te getNextTe(Kyokumen k) {
 92      leaf=node=0;
 93
 94      // 投了にあたるような手で初期化。
 95      Te te=new Te(0,new Position(0,0),new Position(0,0),false);
 96      long time=System.currentTimeMillis();
 97
 98      // 評価値最大の手を得る
 99      negaMax(te,k,-INFINITE,INFINITE,0,DEPTH_MAX);
100
101      time=System.currentTimeMillis()-time;
102 //     System.out.println("leaf="+leaf+" node="+node+" time="+time+"ms");
103
104      return te;
105    }
106 }
```

＜Te.java＞

```
 1 package jp.usapyonsoft.lesserpyon;
 2
 3 public class Te implements Cloneable,Constants {
 4   int koma;                 // どの駒が動いたか
 5   Position from;            // 動く前の位置（持ち駒の場合、０筋０段）
 6   Position to;              // 動いた先の位置
```

```
 7   boolean promote;          // 成る場合、true 成らない場合 false
 8
 9   public Te(int _koma,Position _from,Position _to,boolean _promote) {
10     koma=_koma;
11     from=(Position)_from.clone();
12     to=  (Position)_to.clone();
13     promote=_promote;
14   }
15
16   public boolean equals(Te te) {
17  return (te.koma==koma&&te.from.equals(from)&&te.to.equals(to) && te.promote==promote);
18   }
19
20   public boolean equals(Object _te) {
21     Te te=(Te)_te;
22     if (te==null) return false;
23     return equals(te);
24   }
25
26   public Object clone() {
27     return new Te(koma,from,to,promote);
28   }
29
30   // 手を文字列で表現する。
31   public String toString() {
32     return sujiStr[to.suji]+danStr[to.dan]+
33             Koma.toString(koma)+(promote?"成" "")+
34             (from.suji==0?"打    " "("+sujiStr[from.suji]+danStr[from.dan]+")")+
35             (promote?"" "  ");
36   }
37 }
```