卒業研究報告書

題目

シミュレーション型将棋の開発

指導教員

石水 隆　講師

報告者

16–1–037–0208

永瀬 祐樹

近畿大学理工学部情報学科

令和 2 年 2 月 4 日提出

概要

将棋やチェス, 囲碁などの代表とするボードゲームは二人零話有限確定完全情報ゲームに分類されており, 世界中に様々なバリエーションが存在する.

昨今, 様々なゲームに対してディープラーニングを用いた AI が作られている. とりわけ, 将棋や囲碁ではプロ棋士を凌ぐ AI も現れている. 将棋や囲碁では, プロ棋士達による膨大な棋譜があるため, それを学習データとして用いることができる. 一方, マイナーなゲームや最近作られたゲームでは充分な対戦結果が無いため, 学習データを得ることが難しい.

そこで本研究では従来の将棋とは異なる特徴を持つシミュレーション型将棋を提案し, そのアプリを作成する. 本研究で提案するシミュレーション型将棋を用いることにより, 学習データがない状態から強い AI を開発できるかを最終目標とする.

将棋には持ち駒のルールがあり, それにより類似したゲームであるチェスや将棋には無い奥深さを得ている. そこで, 本研究では，持ち駒のルールをさらに発展させ, ポイントに応じて任意の駒を打てるようにした特殊ルールを加えた将棋を提案する.

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. 本将棋ではプロの棋士により, 駒の評価値はほぼ定まっているため, それを用いることが可能である. 本研究で提案するシミュレーション型将棋は京都将棋をベースに作成した. 京都将棋は「香-と」,「銀-角」,「金-桂」,「飛-歩」と王以外の駒が本将棋と裏表が異なり, 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の駒の性能が変わる点である. また盤面も 5x5 と小さいため, 本将棋の駒の評価値をそのまま使用することができない. そこで, 本研究では, 本研究で提案するシミュレーション型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 本研究の背景

将棋やチェス, 囲碁などの代表とするボードゲームは二人零話有限確定完全情報ゲームに分類されており, 世界中に様々なバリエーションが存在する [1].

二人零話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. 零話とは, 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, お互いのプレイヤーが相手のすべての情報が公開されていることである.

昨今, 様々なゲームに対してディープラーニングを用いた AI が作られている. とりわけ, 将棋や囲碁ではプロ棋士を凌ぐ AI も現れている. 将棋や囲碁では, プロ棋士達による膨大な棋譜があるため, それを学習データとして用いることができる. 一方, マイナーなゲームや最近作られたゲームでは充分な対戦結果が無いため, 学習データを得ることが難しい.

そこで本研究では独自で開発した将棋のアプリを作成する. 作成するにあたって, 学習データがない状態から強い AI を開発できるかを最終目標とする.

将棋には持ち駒のルールがあり, それにより類似したゲームであるチェスや将棋には無い奥深さを得ている. そこで, 本研究では, 持ち駒のルールをさらに発展させ, ポイントに応じて任意の駒を打てるようにした特殊ルールを加えた将棋を提案する.

2 将棋のバリエーション

将棋は様々な派生将棋が存在し, 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 [13] やゴロゴロ将棋 [14] などがある.5 五将棋は 5x5 の盤面と 6 種類の駒, ゴロゴロ将棋は 5x6 の盤面 4 種類の駒を使用している. 他にもどうぶつしょうぎ [6], アンパンマンはじめてしょうぎ [7] などがある.

どうぶつしょうぎとは,2008 年に女流棋士の北尾まどか初段によって考案されたボードゲームである. 将棋に類似しているが, 将棋と比べて非常に簡潔なルールになっている. 盤面 3x4 の 12 マス, 動物絵柄が書かれた駒 8 枚を使用するミニ将棋である. 動物はライオン, 象, キリン, ひよこの 4 種類である. ライオンは本将棋の玉と同じ動き方である. 象は斜めの 4 近傍に動けるが, 本将棋の角と違って隣のマスにしか動けない. キリンは上下左右の 4 近傍に動ける. しかし, 象と同様に隣のマスにしか動けない. ひよこは本将棋の歩と同様に前に 1 マス動ける. 敵陣に入るとにわとりになる. にわとりは斜め後ろ以外の 6 方向に 1 マス動ける. ルールは本将棋とほぼ同様でライオン (玉) を取ったほうが勝つ. さらに, 自陣のライオンが敵陣に入って次の相手の一手で取られなければ勝ちとなる. 同じ局面に 3 回目に到達すると引き分けになる. 将棋の二歩, 打ち歩詰め,1 段目の歩打ち, 王手千日手に該当する反則はない.

アンパンマンはじめてしょうぎ (以下, アンパンマン将棋とする) は,2012 年 6 月 28 日に株式会社セガトイズと北尾まどか女流棋士初段の共同開発により発売されたミニ将棋である. アンパンマン将棋では, 先手をアンパンマンチーム, 後手をばいきんまんチームと呼ばれる. 先手側はアンパンマン, しょくぱんまん, カレーパンマンの 3 種類の駒を, 後手側はばいきんまん, ホラーマン, ドキンちゃんの 3 種類の駒を使用する. 先手後手アンパンマン, ばいきんまんそれぞれリーダーの役割を担う. また, 将棋盤は 3x5 を使用する. アンパンマンとば

いきんまんは斜め下と下方向以外に 1 マス, しょくぱんまんとホラーマンは前方左右に 1 マス, カレーパンマンとドキンちゃんは前方斜め前方に 1 マス進むことが可能である. アンパンマン将棋は本将棋と違い動かしたマスに相手の駒がある場合, そのマスにある相手の駒を盤の外に出すことができるが, 持ち駒にならず, 駒を打つことができない. 勝利条件は, リーダーを取るか, 敵陣にリーダーが入れば勝ちとなる.

5 五将棋やゴロゴロ将棋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完全解析はされていない.

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 [8]. また, アンパンマン将棋は, 双方最善手を指すと千日手で引き分けることが判明している [9].

3 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, モンテカルロ法, 局面の評価値計算, 定石データベース, 一定手先の先読み, 必勝読み, 完全読み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.[12]

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不可能な問題でも, 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. 問題によって他の数値計算手法より簡単に適用できる. しかし, 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.

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ものが用いられている.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成する必要がある.

定石データベースとは, 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. これを使用することでより強い AI になる. しかし, 相手があえて定石以外の手を指すなどの, 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用できない.

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最善の手を打たせることである. 例えば, 先読みの深さを 3 として, 局面の分岐数を x とした場合,3^x 個の局面数を評価し, その中から最善手を打つことが可能である. 一般に先読みする手数が多いほど強い AI となる. しかし, 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.

必勝読みとは, オセロのように勝敗だけでなく石差も問題になるゲームの場合に, 勝敗のみを読み切ることをいう. また, 完全読みとは, 石差までを読み切ることをいう. 必勝読みのほうが計算時間が少なく済むため, 一般的にまず必勝読みで価値を確定させた上で, 残り手数が少なくなると完全読みに切り替えてより点数の高い勝ちを目指すことが多い. 将棋では, 終盤の読みや詰将棋には完全読みが行われている. 将棋のおける完全読みは, 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. 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かけるのかを決めなければならない.

昨今注目されている手法にディープラーニング [10] 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 を作成できる可能性がある. 例えば, 囲碁では, α 碁と呼ばれる AI がプロ棋士に勝つなど目覚ましい成績を上げている [11]. 以上の手法を用いることで, 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成することが可能であ

り, ゲームによってはプロに勝つことも可能である.

3.1 本研究の目的

本研究では,Java を用いて独自の将棋アプリケーションを開発する. 独自で開発するため, まずは人対人の対戦が可能なアプリケーションを作成する.

AI の検証として独自の将棋を用いる理由としては, 今までの既存のミニ将棋だと過去の結果などがあって, その結果に近い答えを出すより新しい将棋を作って学習データが無いところから AI をどこまで強くするかを試みたいからである.

3.2 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 章では本研究の対象である作本研究でシミュレーション型将棋についてのルール等を説明する.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記述する. 4 章では作成したプログラムについて説明する.5 章で検証の詳細, 結果を示す.6 章では, 結論及び今後の課題について述べる.

4 シミュレーション型将棋

本章では, 本研究の対象であるシミュレーション型将棋について説明する.

4.1 京都将棋

本研究では, 新しい将棋を作成にあたって京都将棋をベースとした.[5]

京都将棋は,1976 年に田宮克哉が発表した, ごく新しい将棋である. 京都銀閣将棋, 京都銀閣金鶏秘譜将棋とも言われている.

京都将棋はほぼ本将棋と同様のルールだが, 異なった点がいくつかある. まず盤面が 5x5 と本将棋より小さく自陣, 敵陣の区別はない. そして, 駒が「香-と」,「銀-角」,「金-桂」,「飛-歩」と王以外の駒が本将棋と裏表が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. 初期盤面は以下の図 1 である.

京都将棋の勝利条件や駒の動き方を以下にまとめる.

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能な状態 (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:玉以外の駒は一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. 取った駒を打つ時は, 裏表どちらで打ってもよい.

4.2 シミュレーション型将棋のルール

本研究で提案するシミュレーション型将棋のルールは京都将棋とほぼ同様である. シミュレーション型将棋のルールは以下の通りである.

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

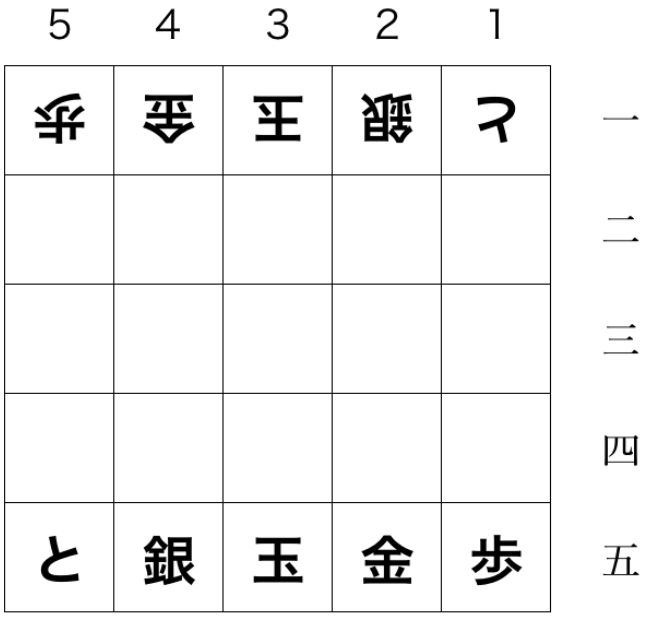

図 1　京都将棋の初期盤面

駒に捕獲されてしまうことが不可能な状態 (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:玉以外の駒は一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. 取った駒を打つ時は, 裏表どちらで打ってもよい.

駒のポイント:初期ポイントは 0pt であり, 一手指すごとに 1 ポイント加算され, 駒を取ると 3 ポイント加算される. 取った駒を打つ時は 10 ポイント消費する必要がある.

今回プログラムを作成するにあたって, 実際に京都将棋を何試合かやった結果, 本将棋と比べて駒の価値が高いことがわかった. 理由としては, 持ち駒が裏表どちらでも打ってよいという制限のないルールは試合の形勢を一気に変えることができる一つの要因と考えられる.

その価値の高いと考えられる持ち駒にポイント機能を追加して制限することで試合がどう変わるのかを検証したいためである.

シミュレーション型と名付けた理由は, 将棋は元々戦場をモチーフにして駒を兵士を見立てたボードゲームであるが, 持ち駒を打つ場合, ノーコストで打てるところが現実の戦場ではないと考えた. そこで今回ポイントをルールとして駒を打つ際にコストをかかる点ではより現実味が溢れると考えた.

一方, 京都将棋は一手毎に駒の裏表が変わるため, 盤上の駒が使いにくいのに対して, 持ち駒は表裏好きな方で打てるため, 盤上の駒と比べて非常に強くなる. そこで本研究では, 本将棋と比べて持ち駒が強過ぎる京都将棋にポイントルールを加えることで持ち駒と盤上の駒の強弱のバランスが得られるか検証する.

5　コンピュータ将棋の着手選択法

本章では, 本研究で作成した将棋 AI の着手選択法について述べる.

5.1　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, 着手可能手から評価値の高い手の選択確率が高くなるようにした上でランダムに選択される. 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かあるか, 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. しかし, 読む深さが深くなるち探索時間は指数的に増え, 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かかる時間と, 深く読むことにより読む局面数が増えることによりか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, 敵味方の盤上の駒及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, プロ棋士たちの長年の研究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, 今回作成した将棋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, 適切な駒の評価値を求める必要である.

5.2　MinMax 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,MinMax 法やその改良である $\alpha\,\beta$ 法が用いられる.

MinMax 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.MinMax 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, 各頂点の子は一手先の局面を示す. 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この値, 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. この操作をすべ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.MinMax 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.MinMax 法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すべて読む必要がある. しかし, この手法は探索時間がかかるという欠点がある.

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, 三手先を読むには, 平均的に「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 [3].

MinMax 法の例を図 2 で示す. 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.

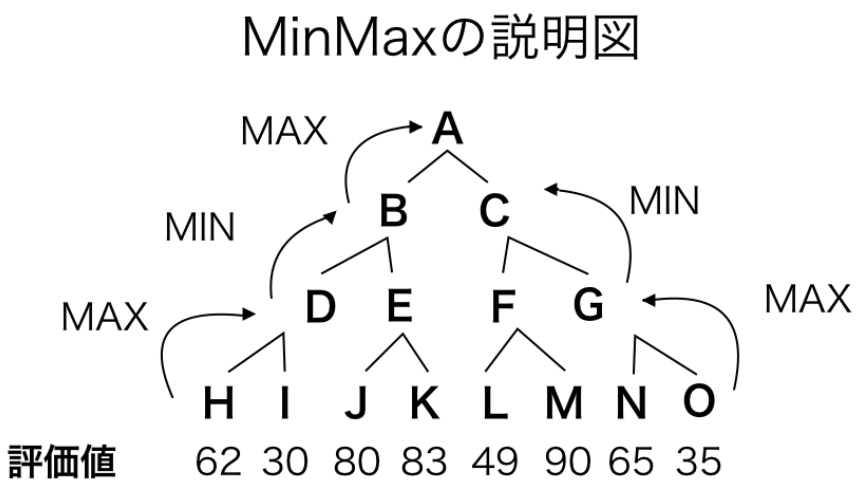

図 2　MinMax 法の例

5.3　αβ法

αβ法とは探索アルゴリズムの一つで MinMax 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,MinMax 法と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 [3].

図 3 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.

6　将棋プログラム

本章では本研究で作成した将棋のアプリケーションについて説明する.

本研究では,[3] の将棋プログラムをベースに,Java を用いて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.

本研究で作成したプログラムは,Constants インターフェイス,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,MainTest クラス,Position クラス,Sikou クラス,Te クラスの 11 個から成る.

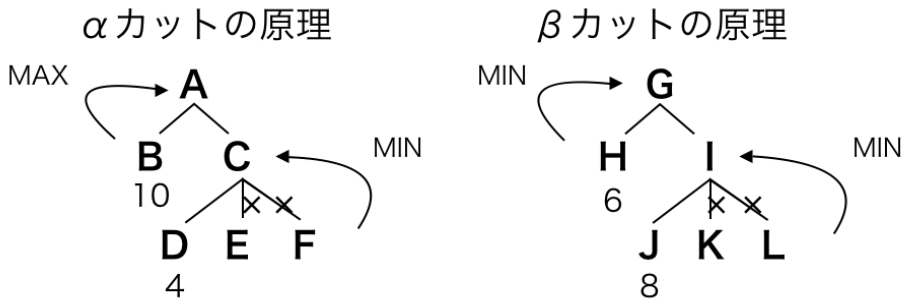

図 3　α カット, β カットの例

6.1　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. 図 4 に Constants インタフェースのクラス図を示す.

<<interface>> Constants	# 定数の定義
+ SENTE : int	# 先手の定義
+ GOTE : int	# 後手の定義
+ sujiStr : String[]	# 筋を表す文字列の定義
+ danStr: String[]	# 段を表す文字列の定義

図 4　Contants インターフェイスのクラス図

6.2　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. 図 5 に KomaMoves インタフェースのクラス図を示す. 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.

<<interface>> KomaMoves	# 駒が動ける方向
+ diffDan: int[]	# 段の移動の定義
+ diffSuji: int[]	# 筋の移動の定義
+ diff : int[]	# 移動の定義
+ canMove: boolean[][]	# 駒が 1 マス動けるかの表
+ canJump: boolean[][]	# 駒が長距離動けるかの表

図 5　KomaMoves インターフェイスのクラス図

6.3 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. 図 6 に Player インターフェイスのクラス図を示す.

<<interface>> Player	#Player の動作を表すクラス
+ getNextTe (k:Kyokumen) : Te	#合法手を生成

図 6　Player インターフェイスのクラス図

6.4 GenerateMoves クラス

GenerateMoves クラスでは, 合法手の生成を行う. 図 7 に GenerateMoves クラスのクラス図を示す. また, 表 1 に各メソッドについてまとめる.

GenerateMoves	# 合法手を生成するクラス
+ removeSelfMate (k:Kyokumen, v:ArrayList<te>) : ArrayList<Te>	# 自殺手を除く
+ addTe (k:Kyokumen, v:ArrayList<te>, teban:int, koma:int, from:int, to:int) : void	# 手を追加する
+ generateLegalMoves (k:Kyokumen) : ArrayList<Te>	# 合法手を生成する

図 7　GenerateMoves クラスのクラス図

表 1　GenerateMoves クラスのメソッド

メソッド	処理内容
removeSelfMate	各手について、自分の玉に王手がかかっていないどうかチェックし, 王手がかかっている手は取り除く
addTe	与えられた ArrayList 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
generateLegalMoves	与えられた局面における合法手を生成する

6.5 Human クラス

Human クラスは人の入力用クラスである. 図 8 に Human クラスのクラス図を示す. 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. 例えば,2 五の金を 2 四に動かして桂に成る手は 2524 と入力する. 駒を打つ時には, 例えば,3 三歩打であれば 0133 と入力する.1 桁目の 0 は打ち駒を表し,2 桁目の数値は駒の種類を表す.3,4 桁目は桁目が駒を打つ位置である. 駒の種類に対応する 2 桁目の数値の対応表を表 2 に示す. 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.

表 2　打ち駒の種類と駒番号の対応

駒	歩	香	桂	銀	金	角	飛	と
駒番号	1	2	3	4	5	6	7	8

Human　　# 人の入力用クラス
+ getNextTe (k:Kyokumen) : Te　　# 合法手を生成

図 8　Human クラスのクラス図

6.6　Koma クラス

Koma クラスは, 駒の種類を定義する. 図 9 に Koma クラスのクラス図を示す. また, 表 3 に各メソッドについてまとめる.

表 3　Koma クラスのメソッド

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

6.7　Kyokumen クラス

Kyokumen クラスは, 盤面や持ち駒など局面を表現する. 図 10 に Kyokumen クラスのクラス図を示す. また, 表 4 に各メソッドについてまとめる.

6.8　MainTest クラス

MainTest クラスは, 実行クラスである. 図 11 にクラスのクラス図を示す. 実行する時に引数を”HUMAN”もしくは”CPU”のいずれかを入力する. 例えば,「HUMAN CPU」と引数を設定すると先手は人が操作, 後手はコンピュータが操作するように実行する.「CPU CPU」の場合コンピュータ同士の対戦となる.

Koma	# 駒の種類を定義するクラス
+ EMPTY:int	# 空
+ EMP:int	# 空の別名
+ PROMOTE:int[]	# 駒の成り先
+ FU : int	# 歩
+ KY : int	# 香
+ KE : int	# 桂
+ GI : int	# 銀
+ KI : int	# 金
+ HI : int	# 飛
+ TO : int	# と
+ GY : int	# 玉
+ SFU : int	# 先手の歩
+ SKY : int	# 先手の香
+ SKE : int	# 先手の桂
+ SGI : int	# 先手の銀
+ SKI : int	# 先手の金
+ SHI : int	# 先手の飛
+ STO : int	# 先手のと
+ SGY : int	# 先手の玉
+ GFU : int	# 後手の歩
+ GKY : int	# 後手の香
+ GKE : int	# 後手の桂
+ GGI : int	# 後手の銀
+ GKI : int	# 後手の金
+ GHI : int	# 後手の飛
+ GTO : int	# 後手のと
+ GGY : int	# 後手の玉
+ WALL : int	# 壁
+ komaString : String[]	# 駒の文字列表現
+ canPromote : boolean[]	# 駒が成れるか
+ isSente (koma:int) : boolean	# 先手の駒か判定
+ isGote (koma:int) : boolean	# 後手の駒か判定
+ isSelf (koma:int) : boolean	# 手番側の駒か判定
+ isEnemy (koma:int) : boolean	# 相手側の駒か判定
+ getKomashu (koma:int) : int	# 駒の種類を得る
+ toBanString (koma:int) : String	# 駒の盤面上の文字列表示
+ toString (koma:int) : String	# 駒の文字列表記
+ toBanString (koma:int) : String	# 駒の盤面表示
+ canPromote (koma:int) : boolean	# 駒が成れるか判定

図 9　Koma クラスのクラス図

Kyokumen	# 盤面や持ち駒など局面を表現クラス
ban:int[]	# 盤面
hand:int[]	# 持ち駒
handP:int[]	# 持ち駒ポイント
teban : int	# 手番
r : Random	# ランダム
eval : int	# 現在の先手から見た評価値
kingS : int	# 先手玉の位置 盤外の利きの届かないところ
kingG : int	# 後手玉の位置 盤外の利きの届かないところ
+ komaValue : int[]	# 駒の評価値
+ Kyokumen	# コンストラクト
+ clone : Kyokumen	# 局面のコピー
+ equals(o : object) : boolean	# 局面が同一かどうか
+ equals(k : kyokumen) : boolean	# 局面が同一かどうか
+ get(p : int) : int	# ある位置にある駒を取得, 盤外なら壁を返す
+ put(p : int,koma : int) : void	# ある位置にある駒を置く
+ turnPt(teban :int) : int	# ターンごとに 1pt 加算
+ move(te : Te) : void	# 与えられた手で行って進めてみる
+ back(te : Te) : void	# 与えられた手で一手戻す
initKingPos() : void	# kingS,kingG を初期化
+ searchGyoku(teban : int) : int	# 玉を探して位置を返す
initEval() : void	# 初期化した際に局面を評価する関数
+ initAll() : void	
+ evaluate() : int	# 局面を評価する関数
csaKomaTbl : String[]	# CSA 形式の棋譜ファイル文字列
+ toString() : String	# 局面を表示

図 10　Kyokumen クラスのクラス図

MainTest	# 実行クラス
SHOW_BOARD : boolean	# 毎回盤面を表示する
player[] : Player	# player[0] が先手が誰か,player[1] が後手が誰か
kyokumenRireki : ArrayList<Kyokumen>	
usage() : void	# 使い方を表示
+ main(argv : String[]) : void	# メインメソッド

図 11　MainTest クラスのクラス図

6.9　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. 図 12 に Position クラスのクラス図を示す. また, 表 5 に各メソッドについてまとめる.

表 4　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	ある位置にある駒を取得する
put	ある位置にある駒を置く
turnPt	一手ごとに 1 ポイントが加算
move	与えられた手で一手進める
back	与えられた手で一手戻す
initKingPos	kingS,kingG を初期化する
searchGyoku	玉の位置を返す
initEval	初期化した際に, 局面を評価する関数
evalute	局面を評価する関数
toString	局面を表示用に文字列化する

Position	# 駒の位置を表すクラス
+ suji : int	# 筋
+ dan : int	# 段
+ Position(_suji : int,_dan : int)	# コンストラクト
+ equals(p * Position) : boolean	# 同一性比較用メソッド
+ equals(o * Object) : boolean	# 同一性比較用メソッド
+ clone() : Object	# コピーを返す
+ add(diffSuji : int,diffDan : int) : void	# ある方向へ動きを行う
+ sub(diffSuji : int,diffDan : int) : void	# ある方向への逆向き動きを行う
+ add(direct : int) : void	# ある方向へ動きを行う
+ sub(direct : int) : void	# ある方向への逆向き動きを行う

図 12　Position クラスのクラス図

6.10　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. 図 13 に Sikou クラスのクラス図を示す. Sikou クラスでは, 手の探索に negaMax 様式を用いた $\alpha\beta$ 法を使用している. $\alpha\beta$ 法で引数で指定された深さ分の先読みを行い, その局面の評価値を計算, 評価値の高い局面になる手の選択確率を高くした上でランダムに手を選択する.

1. 与えられた局面の合法手を生成する
2. 各合法手に対して, 指定された手数先読みして $\alpha\beta$ 法で評価値を求める
3. 評価値順に手をソートする
4. 評価値上位の手の選択確率を高くしてランダムに手を選択する

表 5　Position 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
equals(Object o)	同一性比較用メソッド
clone	局面のコピーを行う
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

Sikou	# コンピュータの思考ルーチン
MUGEN : int	# 無限大
DEPTH_MAX : int	# 読みの深さ
LIMT_DEPTH : int	# 読みの最大の深さ
TRACE : boolean	# 思考中の盤面を表示する
VALUE_CHECK : boolean	# 各手の評価値を表示する
best : Te[][]	# 最善手順を格納する配列
leaf : int	
node : int	
r : Random	
+ Sikou()	# コンストラクト
+ getMax(t : Te,k : Kyokumen,alpha : int,beta : int,depth : int,depthMax : int) : int	
+ getBetter(t : Te,k : Kyokumen,alpha : int,beta : int,depth : int,depthMax : int) : int	# 評価値上位の手を選択する
+ getNextTe(k : Kyokumen) : Te	

図 13　Sikou クラスのクラス図

6.11　Te クラス

Te クラスは手を表現する. 図 14 に Te クラスのクラス図を示す. また, 表 6 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	コンストラクト
equals(Te te)	局面が同一かどうか
equals(Object o)	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

Te	# 手を表現するクラス
koma : int	# どの駒が動いたか
from : int	# 動く前の位置
to : int	# 動いた先に位置
promote : boolean	# 成り
capture : int	# 取った駒
value : int	# 手の評価値
+ Te(_koma : int,_from : int,_to : int,_promote : boolean,_capture : int)	# コンストラクト
+ Te(_koma : int,_from : int,_to : int,_promote : boolean,_capture : int,_value : int)	# コンストラクト
+ equals(te :Te) : boolean	
+ equals(_te :Object) : boolean	
+ clone() : Object	
+ toString() : String	# 手を文字列で表現する
+ getValue() : int	# value のゲッター
+ setValue(_value : int) : void	# value のセッター

図 14　Te クラスのクラス図

7　駒の評価値の検証

本研究で作成した将棋プログラムは, 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, シミュレーション型将棋がベースとしている京都将棋では適切な駒の評価値は定まっていない. そこで本研究では将棋の対戦 AI を作成し, 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, 最適な駒の評価値を求める. 通常の将棋の駒の評価値 (表 7)[3] を裏表で平均を取った CPU(以下を平均 CPU)(表 8) と, 対戦ごとに評価値を変動させる CPU(以下を変動 CPU) の二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.

表 7　本将棋での評価値

歩	香	桂	銀	金	角	飛	と	玉
100	600	700	1000	1200	1800	2000	1200	10000

表 8　平均 CPU の評価値

香と	銀角	金桂	飛歩	玉
900	1400	950	1550	10000

表 9 に対戦結果を示す. 表 9 より, 勝率が一番高くて 6 割で, 後は先手側で 4〜5 割, 後手側で 5〜6 割の勝率の結果となった.

勝率 p の勝負を N 回行った場合の標準偏差は以下の式で与えられる.

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

$p = 0.5$ とした場合，$N = 100$ なら標準偏差は

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

なので, 試行回数 100 回の場合, 危険率 95% の信頼区間は

$$50 \pm \frac{1.96 \cdot 5.0}{100} \cdot 100 = 50 \pm 9.8\%$$

となる. 従って, 試行回数 100 回では勝率 60% を超えないと統計上有意な値とは言えない.

表 9 より, 先手で 5 割を超えているのはあるが 6 割超えたのはほとんどない. 先手で 5 割以上の中で後手で 6 割以上超えているのは香と:600, 銀角:1200, 金桂:600, 飛歩:1200 のときのみである.

8 結論・今後の課題

本研究でポイントを消費して持ち駒を打つシミュレーション型将棋を提案した. また, 人間同士の試合及び対 CPU 戦が可能な将棋プログラムを作成した. 本研究で作成した将棋プログラムの AI は目標にしていた勝率を見出すことはできなかった. しかし, 今回は開発した将棋はあくまで入り口と考えている. 本研究を通して, ポイントの多いプレイヤーが勝ちや打ち駒によってポイントの消費が変わるなど, さらにルールを発展させることも可能であると考えられる.

本研究の改善点としたら, 新しい将棋を開発したと同時に新たな戦略を開発し持たせることによって勝率が安定すると考えている. 評価値を一手ごとに変動したり, 前半と後半で変えていったりと, 試合中に評価値を変動することで戦略も広がり勝率も大きく変わるのではと考えている.

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, この場を借りて感謝を申し上げます.

参考文献

[1] 松田道弘:世界のゲーム事典, 東京出版 (1989)

[2] 山岡忠夫:将棋 AI で学ぶディープラーニング, マイナビ出版 (2018)

[3] 池泰弘:Java 将棋のアルゴリズム, 工学社 (2007)

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

[5] 京都将棋, 株式会社幻冬舎エデュケーション (2014)

[6] どうぶつしょうぎ, 株式会社幻冬舎エデュケーション (2009)

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

[8] 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/

[9] 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

[10] 藤田一弥, 高原歩夢:実装ディープラーニング, オーム社 (2016)

[11] 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,Vol.57,No.4,pp.335-337, 情報処理学会 (2016) http://id.nii.ac.jp/1001/00158059/

[12] 小谷善行, 岸本章宏, 柴原一友, 鈴木豪:ゲーム計算メカニズム―将棋・囲碁・オセロ・チェスのフログラムはどう動く―, コロナ社 (2010)

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

[14]「ごろごろどうぶつしょうぎ」 発売開始！, 日本将棋連盟 https://www.shogi.or.jp/news/2012/11/post_652.html

付録 A　付録

本研究で作成したプログラムのソースファイルを以下の付録にまとめる

- Constants インターフェイス

```
package sotsuken2;
/**
 * 各種定数の定義
 * @author ny38ryu99iomogusa
 *
 */
public interface Constants {
        public final static int SENTE=1<<4;//先手の定義 2^4=16

        public final static int GOTE=1<<5;//後手の定義 2^5=32

        public final static String sujiStr[]={
                "","1","2","3","4","5"
        }; //筋(行)を表す文字列を定義
        public final static String danStr[]={
                        "","一","二","三","四","五"
                }; //段(列)を表す文字列を定義

}
```

- KomaMoves インターフェイス

```
package sotsuken2;

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    12
/*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
 * 添字の1つ目が方向で,2つ目が駒の種類である
 * 香車や飛車,角などの一直線に動く動きについては,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斜め下への動き*/
  {
  //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, true,false,false,false,
//先手の王,と杏圭全　馬龍
false, true,false,false,false,false,false,true,
//空,後手の歩香桂銀金角飛
false,false,false,false, true, true,false,false,
//後手の王,と杏圭全　馬龍
true, true, true, true, true, true,false, true
},
/*方向1真下への動き*/
{
//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, true,false,false,
//先手の王,と杏圭全　馬龍
true, true, true, true, true,false, true,false,
//空,後手の歩香桂銀金角飛
false, true,false,false, true, true,false,false,
//後手の王,と杏圭全　馬龍
true, true, true, true, true,false, true,false
},
/*方向2斜め右下への動き*/
{
//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, true,false,false,false,
//先手の王,と杏圭全 馬龍
false, true,false,false,false,false, true,
//空,後手の歩香桂銀金角飛
```

```
false,false,false,false, true, true,false,false,
//後手の王,と杏圭全 馬龍
true, true, true, true, true, true,false, true
},
/*方向3左への動き*/
{
//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, true,false,false,
//先手の王,と杏圭全　馬龍
true, true, true, true, true,false, true,false,
//空,後手の歩香桂銀金角飛
false,false,false,false,false, true,false,false,
//後手の王,と杏圭全　馬龍
true, true, true, true, true,false, true,false
},
/*方向4右への動き*/
{
//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, true,false,false,
//先手の王,と杏圭全　馬龍
true, true, true, true, true,false, true,false,
//空,後手の歩香桂銀金角飛
false,false,false,false,false, true,false,false,
//後手の王,と杏圭全　馬龍
true, true, true, true, true,false, true,false
},
/*方向5斜め左上への動き*/
{
//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, true, true,false,false,
//先手の王,と杏圭全　馬龍
true, true, true, true, true,false,false, true,
//空,後手の歩香桂銀金角飛
false,false,false,false, true,false,false,false,
//後手の王,と杏圭全　馬龍
false, true,false,false,false,false,false, true
},
/*方向6真上への動き*/
{
//先手でも後手でもない
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, true,false,false, true, true,false,false,
//先手の王,と杏圭全　馬龍
true, true, true, true, true,false, true,false,
//空,後手の歩香桂銀金角飛
false,false,false,false,false, true,false,false,
//後手の王,と杏圭全　馬龍
true, true, true, true, true,false, true,false
},
/*方向7斜め右上への動き*/
{
//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, true, true,false,false,
//先手の王,と杏圭全　馬龍
true, true, true, true, true,false,false, true,
```

```
//空,後手の歩香桂銀金角飛
false,false,false,false, true,false,false,false,
//後手の王,と杏圭全　馬龍
false, true,false,false,false,false,false, true
},
/*方向8先手桂馬飛び*/
{
//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, true,false,false,false,false,
//先手の王,と杏圭全　馬龍
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　馬龍
false,false,false,false,false,false,false,false
},
/*方向9先手桂馬飛び*/
{
//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, true,false,false,false,false,
//先手の王,と杏圭全　馬龍
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　馬龍
false,false,false,false,false,false,false,false
},
/*方向10後手桂馬飛び*/
{
```

```
  //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　馬龍
  false,false,false,false,false,false,false,false,
  //空,後手の歩香桂銀金角飛
  false,false,false, true,false,false,false,false,
  //後手の王,と杏圭全　馬龍
  false,false,false,false,false,false,false,false
  },
  /*方向11後手桂馬飛び*/
  {
  //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　馬龍
  false,false,false,false,false,false,false,false,
  //空,後手の歩香桂銀金角飛
  false,false,false, true,false,false,false,false,
  //後手の王,と杏圭全　馬龍
  false,false,false,false,false,false,false,false
  }
};
/**
 * ある方向にある駒が動けるかどうかを表すテーブル
 * 添字の1つ目が方向で,2つ目が駒の種類である
 * 香車や飛車,角,竜,馬の一直線に動く動きについては,表す
 */
static final public boolean canJump[][]={
  /*方向0斜め下への動き*/
  {
```

```
//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, true,false,
//先手の王,と杏圭全　馬龍
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　馬龍
false,false,false,false,false,false, true,false
},
/*方向1真下への動き*/
{
//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, true,
//先手の王,と杏圭全　馬龍
false,false,false,false,false,false,false, true,
//空,後手の歩香桂銀金角飛
false,false, true,false,false,false,false, true,
//後手の王,と杏圭全　馬龍
false,false,false,false,false,false,false, true
},
/*方向2斜め右下への動き*/
{
//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, true,false,
//先手の王,と杏圭全　馬龍
```

```
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　馬龍
false,false,false,false,false,false, true,false
},
/*方向3左への動き*/
{
//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, true,
//先手の王,と杏圭全　馬龍
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　馬龍
false,false,false,false,false,false,false, true
},
/*方向4右への動き*/
{
//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, true,
//先手の王,と杏圭全　馬龍
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　馬龍
false,false,false,false,false,false,false, true
},
/*方向5斜め左上への動き*/
```

```
{
//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, true,false,
//先手の王,と杏圭全　馬龍
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　馬龍
false,false,false,false,false,false, true,false
},
/*方向6真上への動き*/
{
//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, true,false,false,false,false, true,
//先手の王,と杏圭全　馬龍
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　馬龍
false,false,false,false,false,false,false, true
},
/*方向7斜め右上への動き*/
{
//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, true,false,
```

```
            //先手の王,と杏圭全　馬龍
            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　馬龍
            false,false,false,false,false,false, true,false
            }
            //桂馬の方向に飛ぶ駒はないので省略
          };
  }
```

- Player インターフェイス

```
package sotsuken2;

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

- GenerateMoves クラス

```
package sotsuken2;

import java.util.ArrayList;
/**
 * 合法手
 * @author ny38ryu99iomogusa
 *
 */
public class GenerateMoves implements Constants, KomaMoves {
  /**
  * 各手について,自分の玉に玉手がかかっていないどうかチェックし,
  * 王手がかかっている手は取り除く
  * @param k
  * @param v
  * @return
  */
  public static ArrayList<Te>  removeSelfMate(Kyokumen k,ArrayList<Te>
v){
    ArrayList<Te> removed=new ArrayList<Te> ();
```

```
for(int i=0;i<v.size();i++){
  Te te = v.get(i);//手を取り出す
  //その手で1手進めてみる
  Kyokumen test = (Kyokumen)k.clone();
  test.move(te);
  int gyokuPosition = test.searchGyoku(k.teban);//自玉を探す
  boolean isOuteHouchi=false;//玉手放置しているかどうかフラグ
  //玉周辺12方向から相手の駒が利いていたらその手を取り除く
  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の動けるか
    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removedに追加しない
      isOuteHouchi=true;
      break;
    }
  }
  //玉周辺8方向から相手の駒が利いていたらその手を取り除く
  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了
     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ならチェック終了
     if(Koma.isEnemy(test.teban,koma)) break;
```

```
        }
      }
      if(!isOuteHouchi) removed.add(te);
    }
    return removed;
  }
 /**
  * 与えられたArrayList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
  * @param v
  * @param teban
  * @param koma
  * @param from
  * @param to
  */
  public static void addTe(Kyokumen k,ArrayList<Te> 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
  * @param k
  * @return
  */
  public static ArrayList<Te> generateLegalMoves(Kyokumen k){
    ArrayList<Te> v=new ArrayList<Te>();
    //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];
            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
              //自分の駒だったら次の方向を検討
              if(Koma.isSelf(k.teban, k.get(to))) 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]){
            //そちらに飛ぶことができる
            for(int i=1;i<9;i++){
              //移動先を生成
              int to = from+diff[direct]*i;
              if(k.get(to)==Koma.WALL) break;//行先が盤外だったら行けない
              //自分の駒だったらいかない
              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);
              if(k.get(to)!=Koma.EMPTY) break;//空きマスでなければここで終わり
            }
          }
        }
      }
    }
    /*手番の側の駒を打つ手を生成*/
    for(int i=Koma.FU;i<=Koma.TO;i++){
```

```
        //打つ駒は,手番の側の駒
        int koma = i|k.teban;
        //駒を持っているか
        if(k.hand[koma]>0){
            if ((k.teban==SENTE && k.handP[0]>=10)
            || (k.teban==GOTE && k.handP[1]>=10)) {
              // 持っている。
              int komashu = Koma.getKomashu(koma);
              for(int suji=0x10;suji<=0x50;suji+=0x10){
                for(int dan=1;dan<=5;dan++){
                  int from=0;//移動元
                  int to=suji+dan;//移動先 駒を打つ場所
                  if(k.get(to)!=Koma.EMPTY) continue;
                  Te te=new Te(koma,from,to,false,Koma.EMPTY);
                  //駒の打つ手が可能なことがわかったので合法手に加える
                  v.add(te);
                }
              }
            }
        }
      }
      //自分の玉に王手がかかっている手は取り除く
      v=removeSelfMate(k,v);
      return v;
    }
  }
```

- Human クラス

```
package sotsuken2;

import java.io.*;
import java.util.ArrayList;

/**
 * 人間
 * @author ny38ryu99iomogusa
 *
 */
```

```
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
  //人対人の対戦時に同じ読み込み元を使うため
  static BufferedReader reader=
  new BufferedReader(new InputStreamReader(System.in));
  @Override
  public Te getNextTe(Kyokumen k) {
    //現在の局面での合法手を生成
    ArrayList<Te> v=GenerateMoves.generateLegalMoves(k);
    //返却する手の初期化 投了にあたるような
    //合法手出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 if(k.teban==GOTE) System.out.println("後手番です");
      System.out.print("指し手を入力:");
      //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だったら終了
      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=v.get(i);
          System.out.println(t);
        }
        System.out.println(k);
        continue;
      }
```

```
boolean promote=true;
if(s.length()==5){
 //5文字目が"*"だったら成り
 if(s.substring(4,5).equals("*")) promote=true;
 else {
   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;
}
int koma=0;//駒
//最初の一桁が0の場合,駒打ち
if(fromSuji==0){
  koma=fromDan|k.teban;//駒は手番の側の駒
  fromDan=0;//fromDanをクリア
}
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)){
  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;
      }
  }
```

- Koma クラス

```
  package sotsuken2;
  /**
   * 駒
   * @author ny38ryu99iomogusa
   *
   */
  class Koma implements Constants,Cloneable {
    public static final int EMPTY = 0;//空(から)
    public static final int EMP = EMPTY;//空(から)の別名

    public static final int PROMOTE[]=
     {0,0,0,0,0,0,0,0,0,//先手でも後手でもない駒
      0,0,0,0,0,0,0,0,0,//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
      -6,0,0,0,0,0,0,0,0,//先手の王,と杏圭全　馬龍
      0,6,6,2,2,-2,-2,-6//空,後手の歩香桂銀金角飛
      -6,0,0,0,0,0,0,0,0//後手の王,と杏圭全　馬龍
     };

     public static final int FU = 1;//歩
     public static final int KY = 2;//香
     public static final int KE = 3;//桂
     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
     public static final int TO = 8;//と
     public static final int GY = 9;//玉
```

```
/*先手*/
public static final int SFU = SENTE+FU;//歩
public static final int SKY = SENTE+KY;//香
public static final int SKE = SENTE+KE;//桂
public static final int SGI = SENTE+GI;//銀
public static final int SKI = SENTE+KI;//金
public static final int SKA = SENTE+KA;//角
public static final int SHI = SENTE+HI;//飛
public static final int STO = SENTE+TO;//と
public static final int SGY = SENTE+GY;//玉
/*後手*/
public static final int GFU = GOTE+FU;//歩
public static final int GKY = GOTE+KY;//香
public static final int GKE = GOTE+KE;//桂
public static final int GGI = GOTE+GI;//銀
public static final int GKI = GOTE+KI;//金
public static final int GKA = GOTE+KA;//角
public static final int GHI = GOTE+HI;//飛
public static final int GTO = GOTE+TO;//と
public static final int GGY = GOTE+GY;//玉

public static final int WALL=64;//盤の外を表すための定数
/**
 * 先手の駒かどうか判定
 * @param koma
 * @return
 */
 static public boolean isSente(int koma){
   return (koma&SENTE)!=0;
 }
 /**
  * 先手の駒かどうか判定
  * @param koma
  * @return
  */
  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
 * @param teban
 * @param koma
 * @return
 */
 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
  * @param teban
  * @param koma
  * @return
  */
  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
   * 先手後手のフラグをビット演算でなくせば良い
   * @param koma
   * @return
   */
   static public int getKomashu(int koma){
     return koma & 0x0f;
   }
   //駒の文字列
   static public final String komaString[]={
     "  ","歩","香","桂","銀","金","角","飛",
     "と","玉","杏","圭","全","  ","馬","竜"
   };
   /**
```

```
 * 駒の文字列化 盤面表示
 * @param koma
 * @return
 */
 static public String toBanString(int koma){
   if(koma==EMPTY) return "    ";
   //先手駒の頭に↑つける
   else if((koma&SENTE)!=0) return "↑"+komaString[getKomashu(koma)];
   //後手駒の頭に↓つける
   else return "↓"+komaString[getKomashu(koma)];
 }
/**
 * 駒の文字列化  持ち駒、手など表示
 * @param koma
 * @return
 */
 static public String toString(int koma){
   return komaString[getKomashu(koma)];
 }
 //駒が成れるかどうか表す
 public static final boolean canPromote[]={
   //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, true, true, true, true,false, true, true,
   //先手の王,と杏圭全  馬龍
   false,false,false,false,false,false,false,false,
   //空,後手の歩香桂銀金角飛
   false, true, true, true, true,false, true, true,
   //後手の王,と杏圭全  馬龍
   false,false,false,false,false,false,false,false
 };
/**
 *
 * @param koma
```

```
     * @return
     */
     static public boolean canPromote(int koma){
       return canPromote[koma];
     }
}
```

- Kyokumen クラス

```
package sotsuken2;

import java.util.Random;
/**
 *
 * @author ny38ryu99iomogusa
 *
 */
class Kyokumen implements Constants,Cloneable {
  int ban[];//盤面
  int hand[];//持ち駒
  int handP[];//持ち駒ポイント
  int teban=SENTE;//手番
  Random r = new Random();
  int eval=0;///現在の先手から見た評価値
  //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;
  //駒の評価値 本将棋の評価値から表と裏の平均を取った
  /*final static int komaValue[]={
    0,0,0,0,0,0,0,0, //何のない場所及び
    0,0,0,0,0,0,0,0, //先手後手でもない駒
    0,1550,900,950,1400,950,1400,1550, //何もない場所,先手の歩,香,桂,銀,金,角,飛
    900,10000,0,0,0,0,0,0, //先手玉及びと~竜
    //何もない場所,後手の歩,香,桂,銀,金,角,飛
    0,-1200,-2200,-1200,-600,-1200,-600,-1200,
    -2200,-10000,0,0,0,0,0,0 //後手玉及びと~竜
  };*/
  final static int komaValue[]={
```

```
    0,0,0,0,0,0,0,0,0, //何のない場所及び
    0,0,0,0,0,0,0,0,0, //先手後手でもない駒
    //何もない場所,先手の歩,香,桂,銀,金,角,飛
    0,1200,2200,1200,600,1200,600,1200,
    2200,10000,0,0,0,0,0,0,0, //先手玉及びと~竜
    //何もない場所,後手の歩,香,桂,銀,金,角,飛
    0,-1550,-900,-950,-1400,-950,-1400,-1550,
    -900,-10000,0,0,0,0,0,0,0 //先手玉及びと~竜
};

public Kyokumen(){
    ban = new int[16*11];
    hand = new int[Koma.GTO+1]; //32+8+1=41
    handP = new int[2];
    handP[0] = 0;
    handP[1] = 0;
    //盤面全体を壁で一旦埋める
    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
    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
 */
 public Kyokumen 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];
  }
  k.teban = teban;//手番のコピー
  k.eval = eval;//評価値のコピー
  k.kingS=kingS;//玉の位置のコピー
  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
 * @param k
 * @return
 */
 public boolean equals(Kyokumen k){
   if(teban!=k.teban) return false;//手番の比較
   /*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が,比較対象の盤面上の同じどうか比較*/
       if(ban[suji+dan]!=k.ban[suji+dan]) return false; //違っていたらfalse
     }
   }


   /**
    * 持ち駒の比較
    */
```

```
    for(int i=Koma.SFU;i<Koma.GTO;i++){
      if(hand[i]!=k.hand[i]) return false;
    }
    // 持ち駒ポイントの比較
    if (handP[0] != k.handP[0]) return false;
    if (handP[1] != k.handP[1]) return false;
    return true;
}
/**
 * ある位置にある駒を取得
 * 盤外なら壁を返す
 * @param pos
 * @return
 */
 public int get(int p){
   if(p<0||16*11<p) return Koma.WALL;
   return ban[p];
 }
 /**
  * ある位置にある駒を置く
  * @param p
  * @param koma
  */
  public void put(int p,int koma){
    ban[p]=koma;
  }
  /**
   * ターンごとに1pt加算
   * @param teban
   */
   public int turnPt(int teban){
     if(teban==SENTE) return handP[0]++;
     else return handP[1]++;
   }
   /**
    * 与えられた手で行って進めてみる
    * @param te
```

```
*/
public void move(Te te){
  if(get(te.to)!=Koma.EMPTY){//駒の行き先に駒があったなら
    eval-=komaValue[get(te.to)];
    /if(teban==SENTE) eval-=komaValue[get(te.to)];
    //else eval-=komaValue1[get(te.to)];;
    //持ち駒にする
    if(Koma.isSente(get(te.to))){
      int koma = get(te.to);//取った駒が先手の駒なら後手の持ち駒に
      //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;//成りなどのフラグ,先手後手の駒のフラグをクリア
      //koma = koma|GOTE;//後手の駒としてフラグをセット
      hand[koma]++;//持ち駒に追加
      //後手の持ち駒ポイント追加
      handP[1]+=3;
      eval+=(3*komaValue[koma])/2;//持ち駒の価値1.5倍
    } else{
      int koma = get(te.to);//取った駒が後手の駒なら先手の持ち駒に
      //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;//成りなどのフラグ,先手後手の駒のフラグをクリア
      //koma = koma|SENTE;//先手の駒としてフラグをセット
      hand[koma]++;//持ち駒に追加
      //先手の持ち駒ポイント追加
      handP[0]+=3;
      eval+=(3*komaValue[koma])/2;//持ち駒は1.5倍の価
      //eval+=(3*komaValue1[koma])/2;//持ち駒は1.5倍の価値
    }
```

```
    }
    //持ち駒を打った.先手の駒なら先手の持ち駒を減らす.ポイント10pt消費
    if(te.from==0){
      hand[te.koma]--;
      //持ち駒ポイントを減らす
      if(Koma.isSente(te.koma)) handP[0]-=10;
      else handP[1]-=10;
      eval-=(3*komaValue[te.koma])/2;///持ち駒は1.5倍の価値
    } else put(te.from,Koma.EMPTY);//盤上の駒を進めた　元の位置はEMPTYに
    //駒を移動先に進める
    int koma=te.koma;
    if(te.promote) koma=koma+Koma.PROMOTE[koma]; //成りの処理
    put(te.to,koma);
    if(te.koma==Koma.SGY) kingS=te.to;
    else if(te.koma==Koma.GGY) kingG=te.to;
  }
  /**
   * 与えられた手で一手戻す
   * @param te
   */
   public void back(Te te){
     put(te.to,te.capture);//散った駒を盤に戻す
     eval+=komaValue[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;//成りなどのフラグ,先手後手の駒のフラグをクリア
         //koma=koma|GOTE;//後手の駒としてのフラグをセット
```

```
      hand[koma]--;//持ち駒に追加
      //後手の持ち駒ポイント減少
      handP[1]-=3;
      eval-=(3*komaValue[koma])/2;// 持ち駒は1.5倍の価値
    } else{
      //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.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;//成りなどのフラグ,先手後手の駒のフラグをクリア
      //koma=koma|SENTE;//先手の駒としてのフラグをセット
      hand[koma]--;//持ち駒に追加
      //先手の持ち駒ポイント減少
      handP[0]-=3;
      eval-=(3*komaValue[koma])/2;// 持ち駒は1.5倍の価値
      //eval-=(3*komaValue1[koma])/2;// 持ち駒は1.5倍の価値
    }
  }
  if(te.from==0){
    //駒打ちだったので持ち駒に戻す
    hand[te.koma]++;
    //駒打ちだったので、持ち駒ポイントに戻す
    if (Koma.isSente (te.koma)) handP[0]+=10;
    else handP[1]+=10;
    eval+=(3*komaValue[te.koma])/2; //持ち駒は1.5倍の価値
  } else {
    //動かした駒を元の位置に戻す
    put(te.from,te.koma);
    if(te.promote){
      //成った後の駒の価値を減じる
      int koma=te.koma+Koma.PROMOTE[te.koma];
      eval-=komaValue[koma];
      //成る前の駒の価値を減じる
```

```
        eval+=komaValue[te.koma];
      }
    }
    if(te.koma==Koma.SGY) kingS=te.from;
    else if(te.koma==Koma.GGY) kingG=te.from;
}
/**
 * kingS,kingGを初期化
 */
 void initKingPos(){
    //先手,後手の玉の位置 盤外の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.SGY) kingS=suji+dan;
        if(ban[suji+dan]==Koma.GGY) kingS=suji+dan;
      }
    }
 }
/**
 * 玉を探して位置を返す
 * @param teban
 * @return
 */
 public int searchGyoku(int teban){
    if(teban==SENTE) return kingS;
    else return kingG;
 }
/**
 * 初期化した際に局面を評価する関数
 */
 void initEval(){
    eval=0;
    /*盤面の駒*/
```

```
    for(int suji=0x10;suji<0x50;suji+=0x10){
      for(int dan=1;dan<=5;dan++){
        eval+=komaValue[ban[suji+dan]];
      }
    }
    /*持ち駒*/
    /*for(int i=Koma.SFU;i<=Koma.STO;i++){
      eval+=komaValue[i]+hand[i];
    }
    for(int i=Koma.GFU;i<=Koma.GTO;i++){
      eval+=komaValue[i]+hand[i];
    }*/
    //次に、持ち駒ポイント
    eval+=(3*handP[0])/2;//持ち駒は1.5倍の価値
    eval-=(3*handP[1])/2;//持ち駒は1.5倍の価値
  }
  public void initAll(){
    initEval();
    initKingPos();
  }
  /**
   * 局面を評価する関数
   * @return
   */
   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",
     "GY","TO","NY","NK","ZE","  ","UM","RY",
     "  ","+FU","+KY","+KE","+GI","+KI","+KA","+HI",
     "+GY","+TO","+NY","+NK","+ZE","  ","+UM","+RY",
     "  ","-FU","-KY","-KE","-GI","-KI","-KA","-HI",
     "GY","-TO","-NY","-NK","-ZE","  ","-UM","-RY",
   };
```

```
/**
 * 局面を表示
 */
 public String toString(){
   String s="";
   //後手持ち駒
   s+="後手持ち駒ポイント:";
   s += handP[1];
   s+="\n";
   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
   s+="先手持ち駒ポイント:";
   s += handP[0];
   s+="\n";
   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;
          }
}
```

- MainTest クラス

```
package sotsuken2;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.util.ArrayList;

public class MainTest implements Constants{
  //初期盤面を与える
  static final int ShokiBan[][]={
    {Koma.GTO,Koma.GKI,Koma.GGY,Koma.GGI,Koma.GFU},
    {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.SFU,Koma.SGI,Koma.SGY,Koma.SKI,Koma.STO}
  };

  static final boolean SHOW_BOARD = false; // true にすると毎回盤面を表示する
  //player[0]が先手が誰か,player[1]が後手が誰か
  static Player player[]=new Player[2];
  static ArrayList<Kyokumen> kyokumenRireki=new ArrayList<Kyokumen>();

  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");
    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");
  }
```

```
public static void main(String[] argv) {
  //先手番後手番が引数で与えられてるか
  if(argv.length<2) {
    System.out.println("error");
    return; //引数が足りない：終了
  }
  //先手番が誰か設定
  if(argv[0].equals("HUMAN")) player[0]=new Human();
  else if(argv[0].equals("CPU")) player[0]=new Sikou();
  else {
    usage();
    return;//引数がおかしい：終了
  }
  //後手番が誰か設定
  if(argv[1].equals("HUMAN")) player[1]=new Human();
  else if(argv[1].equals("CPU")) player[1]=new Sikou();
  else {
    usage();
    return;//引数がおかしい：終了
  }

  try{
    Kyokumen k=new Kyokumen();
    if(argv.length==2){//引数の指定がない場合は初期配置
      k.teban=SENTE;//先手番
      for(int dan=1;dan<=5;dan++){
        for(int suji=5;suji>=1;suji--){
          k.ban[(suji<<4)+dan]=ShokiBan[dan-1][5-suji];
        }
      }
      k.initAll();//諸々の初期化
    } 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み
      String csaFileName=argv[2];
      File f=new File(csaFileName);
      BufferedReader in=new BufferedReader(new FileReader(f));
      ArrayList<String> v=new ArrayList<String>();
```

```
    String s;
    while((s=in.readLine())!=null){
      System.out.println("Read:"+s);
      v.add(s);
    }
    String csaKifu[]=new String[v.size()];
    //v.copyInto(csaKifu);
  }
  int tesu=0;
  //対戦のメインループ
  while(true){
    tesu++;
    System.out.println(tesu+"ターン");
    if(k.teban==SENTE) System.out.println("【先手】");
    else System.out.println("【後手】");
    if(SHOW_BOARD) System.out.println (k);
    //現在の局面を局面の履歴に保存
    kyokumenRireki.add(k.clone());
    //現在の局面での合法手を生成
    ArrayList<Te> v=GenerateMoves.generateLegalMoves(k);
    if(v.size()==0){
      //手番側の負け
      if(k.teban==SENTE) {System.out.println("後手の勝ち");}
      else {System.out.println("先手の勝ち"); }
      break;//対局終了
    }
    //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 出てきた回数を増やす
      if(kyokumenRireki.get(i).equals(k)) sameKyokumen++;
    }
    if(sameKyokumen>=4){
      //同一局面4回以上の繰り返し:千日手
      System.out.println("千日手です");
      break;
```

```
        }
        //指された手を表示
        System.out.println(k.toString());
        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());
        //合法手出ない手をさした場合即負け
        if(!v.contains(te)){
          System.out.println("合法手でない手で指されました");
          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");
          break;
        }
        k.move(te);//指された手で局面を進める
        k.turnPt(k.teban);
        //moveでは、手番が変わらないので局面の手番を変更
        if(k.teban==SENTE) k.teban=GOTE;
        else k.teban=SENTE;
      }
      //対局終了
      System.out.println("対局終了");
      System.out.println(k.toString());
    } catch(Exception ex){
      ex.printStackTrace();
    }
  }
}
```

- Position クラス

```
package sotsuken2;

public class Position implements Cloneable, KomaMoves {
  public int suji;//筋
```

```
public int dan;//段
public Position(){
  suji=0;
  dan=0;
}
public Position(int _suji,int _dan){
  suji=_suji;
  dan=_dan;
}
/**
 * 同一性比較用メソッド
 * @param p
 * @return
 */
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);
 }
 /**
  * ある方向へ動きを行う
  * @param diffSuji
  * @param diffDan
  */
  public void add(int diffSuji,int diffDan){
    suji+=diffSuji;
    dan+=diffDan;
  }
```

```
    /**
     * ある方向への逆向き動きを行う
     * @param diffSuji
     * @param diffDan
     */
     public void sub(int diffSuji,int diffDan){
       suji-=diffSuji;
       dan-=diffDan;
     }
     /**
      * ある方向へ動きを行う
      * @param direct
      */
      public void add(int direct){
        add(diffSuji[direct],diffDan[direct]);
      }
     /**
      * ある方向への逆向きへの動きを行う
      * @param direct
      */
      public void sub(int direct){
        sub(diffSuji[direct],diffDan[direct]);
      }
}
```

- Sikou クラス

```
package sotsuken2;

import java.util.ArrayList;
import java.util.Random;
/**
 * コンピュータの思考ルーチン
 * @author ny38ryu99iomogusa
 *
 */
public class Sikou implements Player,Constants {
  //無限大を表す
  static final int MUGEN=99999999;
```

```
//読みの深さ
static final int DEPTH_MAX=4;
//読みの最大の深さ これ以上は不可能
static final int LIMIT_DEPTH=16;
static final boolean TRACE = false; // true にすると思考中の盤面を表示する
static final boolean VALUE_CHECK = false; // true にすると各手の評価値を表示する
//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;
Random r;
public Sikou(){ r=new Random(); }

int getMax(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
  ArrayList<Te>  v=GenerateMoves.generateLegalMoves(k);
  //現在の指し手の候補手の評価値を入れる
  int value = -MUGEN;
  //合法手の中から一手指してみて1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=v.get(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から
    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
    Te tmpTe=new Te(0,0,0,false,0);
    int eval=-getMax(tmpTe,k,-beta,-alpha,depth+1,depthMax);
```

```
      //int eval=getMin(tmpTe,k,-beta,-alpha,depth+1,depthMax);
      k.back(te);
      //backでは,先手後手を入れ替えないから
      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) {
        value=eval;//返す値更新
        if(eval>alpha) alpha=eval; //α値更新

        //最善手更新
        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;
  }
  /*int getMin(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 = MUGEN;
    //合法手の中から一手指してみて1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から
      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
      Te tmpTe=new Te(0,0,0,false,0);
      int eval=getMax(tmpTe,k,beta,alpha,depth+1,depthMax);
      k.back(te);
      //backでは,先手後手を入れ替えないから
      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) {
        value=eval;//返す値更新
        if(eval<beta) beta=eval; //β値更新
        //最善手更新
        best[depth][depth]=te;
        t.koma=te.koma;
        t.from=te.from;
        t.to=te.to;
        t.promote=te.promote;
        //最善手順更新
        for(int j=depth+1;j<depthMax;j++){
          best[depth][j]=best[depth+1][j];
        }
        //αカットの条件を満たしていたらループ終了
        if(eval<=alpha) break;
      }
    }
    return value;
  }
```

```
*/

/**
 * 評価値上位の手を選択する
 * 各合法手に対して，評価値を決定し，評価値順に並べる．
 * その後，評価値上位の手が選択される確率を高くした上でランダムに手を選択する．
 */
 int getBetter (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評価値を返して終了
   if(depth>=depthMax){
     leaf++;
     if(k.teban==SENTE) return k.evaluate();
     else return -k.evaluate();
   }
   if (TRACE) {
     if (k.teban==SENTE) System.out.println("先手");
     else System.out.println("後手");
     System.out.println("depth "+depth);
     System.out.println(k);
   }
   node++;
   //現在の局面での合法手を生成
   ArrayList<Te> v=GenerateMoves.generateLegalMoves(k);
   //現在の指し手の候補手の評価値を入れる
   int value=-MUGEN;

   /* 合法手の中から一手指してみて各手の評価値を求める */
   for(int i=0;i<v.size();i++){
     //合法手を取り出す
     Te te=v.get(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から
     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
     Te tmpTe=new Te(0,0,0,false,0);
```

```
    int eval=-getMax(tmpTe,k,-beta,-alpha,depth+1,depthMax);
    te.setValue(eval);
    k.back(te);
    //backでは,先手後手を入れ替えないから
    if(k.teban==SENTE) k.teban=GOTE;
    else k.teban=SENTE;
  }

  /* 合法手を評価値順に並べなおす */
  for(int i=0; i<v.size()-1; ++i){
    int bestValue=v.get(i).getValue();
    int bestPlace=i;
    for(int j=i+1;j<v.size();++j){
      int currentValue=v.get(j).getValue();
      if(currentValue>bestValue){
        bestValue = currentValue;
        bestPlace = j;
      }
    }
    if (bestPlace != i) {
      Te te = v.get(bestPlace);
      v.remove(bestPlace);
      v.add(i, te);
    }
  }
  if(VALUE_CHECK){
    for(int i=0;i<v.size();++i){
      Te te=v.get(i);
      System.out.println(te+" : "+te.getValue());
    }
  }

  /* 評価値上位の手の選択確率を高くしてランダムに手を選択する */
  int r1 = r.nextInt(v.size());
  int r2 = r.nextInt(v.size());
  int betterPlace = (r1<r2) ? r1:r2;
  Te te = v.get(betterPlace);
```

```
      value = te.getValue();
      best[depth][depth]=te;
      t.koma=te.koma;
      t.from=te.from;
      t.to=te.to;
      t.promote=te.promote;
      //最善手順更新
      for(int j=depth+1;j<depthMax;j++){
        best[depth][j]=best[depth+1][j];
      }
      value += 3*v.size()/2;
      return value;
    }
    @Override
    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 d = r.nextInt(DEPTH\_MAX-1)+1;
      //int v=getMax(te,k,-MUGEN,MUGEN,0,d);
      //int v=getMax(te,k,-MUGEN,MUGEN,0,DEPTH\_MAX);
      //int v1=getMin(te,k,-MUGEN,MUGEN,0,DEPTH\_MAX);
      int v=getBetter(te,k,-MUGEN,MUGEN,0,DEPTH\_MAX);
      System.out.println("先読みの評価値:"+v);
      //System.out.println("min先読みの評価値:"+v1);
      System.out.print("最善手順:");
      for(int i=0;i<DEPTH\_MAX;i++){
        System.out.print(best[0][1]+"。");
      }
      System.out.println();
      time=System.currentTimeMillis()-time;
      System.out.println("leaf="+leaf+" node="+node+" time"+time+"ms");
      return te;
    }
}
```

- Te クラス

```
package sotsuken2;

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筋0段)
  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
  int value; // 手の評価値

  public Te(int _koma,int _from,int _to,boolean _promote,int _capture){
    koma=_koma;
    from=_from;
    to=_to;
    promote=_promote;
    capture=_capture;
    value=0;
  }
  public Te(int _koma,int _from,int _to,boolean _promote,
            int _capture, int _value){
    koma=_koma;
    from=_from;
    to=_to;
    promote=_promote;
    capture=_capture;
    value=_value;
  }

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

  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?"":"   ");
   }
   /**
    * valueのgetter
    * @return value
    */
    public int getValue() {
      return value;
    }
    /**
     * valueのsetter
     * @param _value value
     */
     public void setValue (int _value) {
       value = _value;
     }
}
```

表 9　検証結果

変動 CPU の評価値					変動 CPU が先手側				変動 CPU が後手側			
香と	銀角	金桂	飛歩	玉	勝	負	分	勝率	勝	負	分	勝率
1200	1700	1200	2200	10000	48	52	-	48	63	37	-	63
600	2000	500	2000	10000	34	66	-	34	63	37	-	63
1000	1200	900	1200	10000	54	45	-	54	56	44	-	56
1300	1300	1000	1300	10000	36	64	-	36	66	34	-	66
1300	2000	1300	2200	10000	43	57	-	43	59	41	-	59
1000	1000	500	1000	10000	46	56	-	46	63	37	-	63
2200	1200	1200	1200	10000	48	52	-	48	54	46	-	54
600	600	1200	600	10000	50	50	-	50	54	46	-	54
1200	600	600	600	10000	52	48	-	52	57	43	-	57
600	600	600	1200	10000	45	55	-	44	58	42	-	58
600	1200	600	600	10000	55	45	-	55	59	41	-	59
1200	1200	600	600	10000	56	44	-	56	58	42	-	58
1200	600	1200	600	10000	49	51	-	49	63	37	-	63
1200	600	600	1200	10000	38	62	-	38	55	45	-	55
600	1200	600	1200	10000	50	50	-	50	66	34	-	66
600	1200	1200	600	10000	45	55	-	45	61	39	-	61
600	600	1200	1200	10000	47	53	-	47	60	40	-	60
600	1200	1200	1200	10000	51	49	-	51	51	49	-	51
1200	600	1200	1200	10000	51	49	-	51	55	45	-	55
1200	1200	600	1200	10000	45	55	-	44	58	42	-	58
1200	1200	1200	600	10000	48	52	-	48	63	37	-	63
600	1700	600	600	10000	40	60	-	40	53	47	-	53
1200	200	1200	1200	10000	40	60	-	40	69	31	-	69
1200	2200	1200	1200	10000	41	59	-	41	62	38	-	62
1200	1200	2200	1200	10000	49	51	-	49	52	48	-	52
1200	1200	1200	2200	10000	41	59	-	41	57	43	-	57
1200	1700	1200	2200	10000	34	66	-	34	63	37	-	63
1200	600	1200	2200	10000	55	45	-	55	58	42	-	58
1200	600	1200	2200	10000	34	66	-	34	63	37	-	63
2200	600	1200	2200	10000	48	52	-	48	55	45	-	55
2200	600	1200	1200	10000	35	65	-	35	65	35	-	65