卒業研究報告書

題目

将棋によるライバル AI の作成

指導教員

石水 隆 講師

報告者

14–1–037–0095

星宮 涼太

近畿大学理工学部情報学科

平成 30 年 1 月 23 日提出

概要

本研究は, 将棋において対戦相手と対戦の中で同じような実力に調整することで対戦相手にとってライバルになれるような AI, 通称ライバル AI を作成し, そのライバル AI の性能を検査するライバル AI は対戦相手の実力をチェックするために盤面の評価値をだし, その高さによって自身の棋力を変化させる. 本研究では java 言語でライバル AI を作成する. 本研究で作成するライバル AI は, 対戦相手の指した手の評価値の高さによって, 自身の指す手を変える AI である. ライバル AI は, 各候補手に評価値を設定し, 相手が評価値の高い手を指せば自分も評価値の高い手を指し, 評価値の低い手ならば自分も評価値の低い手を選択する事により, 相手と同程度の強さになるようにする. 候補手の評価方法は, 数手先の局面を先読みし, その局面の評価値を元に $\alpha\,\beta$ 法を用いて算出する.

目次

1 序論

1.1 本研究の背景

将棋の AI は日々進化しており, 昨今では計算機の性能や探索アルゴリズムの発展と向上, またディープラーニング [5] の出現によって人間のトッププレーヤーにも負けないような強い将棋 AI を作成することが可能なレベルに達している. 将棋では佐藤天彦名人が「PONANZA」勝利して以来,AI は将棋協会が AI とプロ棋士との対戦を将棋協会が規制するほど AI は強くなってきている [7]. そのため, 多くの人間が将棋 AI と戦う場合は強さのレベルを下げて戦うということになる. しかし, ここで自分の実力に合わせた適正なレベルに下げないと, 強すぎて手も足も出ずに全く勝てなかったり, 逆に弱すぎてあっさり勝ってしまう可能性がある.

1.2 本研究の目的

1.1 節で述べたように人間が AI を相手に将棋を楽しむには AI をプレイヤーの実力にあった適正なレベルを設定する必要がある. しかし, 適正なレベルに調整するためには将棋 AI と人間が数局対戦し, 自分のレベルを探す必要があり, 適正なレベルの将棋 AI と対戦するのに時間がかかってしまう. そこで本研究では対戦中に対戦相手のレベルに自身のレベルが近くなるように自動調整し, 対戦相手と同じ実力になるような AI, 通称ライバル AI を作成し, そのライバル AI の性能を検査する.

1.3 将棋に関する既知の結果

本節では将棋に関する既知の結果を示す

1.3.1 着手選択方法

本小節では, 各局面で可能な合法手の中から選択する方法について述べる. 一般的に, 将棋のプログラムは,「評価関数」と「先読み」を用いている. 評価関数とは, 今回でいうと盤面がどちらがどの程度有利なのかを判断する関数である. 評価関数が局面の評価（有利不利）を間違えるようなら, プログラムは相手の実力を間違えて認識していまう. したがって, 思考ゲーム・プログラムでは評価関数が非常に重要になる. 将棋の場合, 各駒に点数を付けその合計を評価値とする, という手法がよく用いられている. 例えば, 歩 100 点, 香 600 点, 桂 700 点,... とし, 先手が歩 1 個得している局面なら先手が +100 点, 後手が-100 点で差し引き先手が 200 点有利と評価する. 他にも, 各駒の動けるマス目の数, 駒同士の連携, 玉の固さ, 等様々な評価基準がある. ある局面において評価関数が高い場合でも, そこから何手か進むと大駒を取られたり不利な駒配置になったりして不利になってしまうこともある. これを避けるためには, 評価関数を現在の局面ではなく, 数手先に現れる局面を使って求める必要がある. これを先読みと言う. 先読みをする方法として探索アルゴリズムのミニマックス法や, それの改良版である $\alpha\,\beta$ 法が用いられている. ミニマックス法については 2.2 節にて, $\alpha\,\beta$ 法については 2.3 節にて説明する.

1.3.2 着手をするための既知の手法

はじめに既知の手法として駒の関係を利用した評価関数がある [9]. これは駒の損得で評価値をだすだけの手法に駒の関係を評価項目として加える試みである. 盤面にある二つの駒 (q,p) にたいしてその二つの駒の位置,

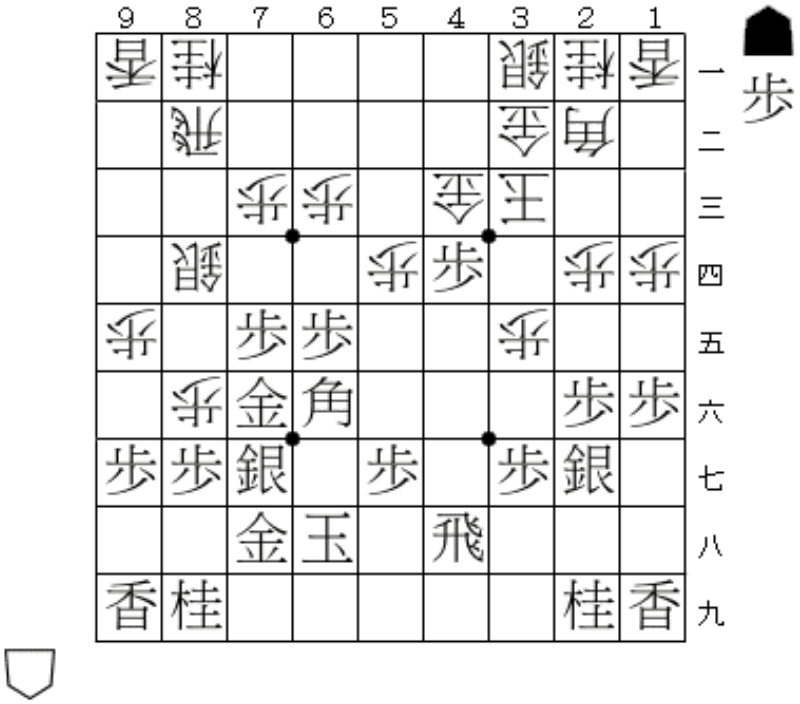

図 1　評価の例

種類, 先手後手の三つの情報から評価しする方法である. 次にモンテカルロ木探索を将棋へと適用した評価関数である [8]. その方法には, 読みの深さが一定の深さに達した時に評価関数の値から勝敗を定める方法と深さに関係なく評価値が閾値を超えた時に勝敗を返す方法がある. このように昨今ではいろいろな手法が試されている.

1.3.3　ライバル AI

ライバル AI とは, 相手の実力に合わせて自分も実力を合わせる AI のことをさす. ゲームをする上で, 実力が同じ者同士の勝負は面白く, ゲームの上達にもつながる. 従って, プレイヤーと同程度の実力を示すライバル AI は不自然ではない手でかつ弱い手を指せることが求められる. 上田は遺伝アルゴリズムを用いてオセロに対するライバル AI を構成する手法を示した [3]. 次の三つの要素, プレイヤの技術の計測, 模倣. 局面評価から自然な着手に着目してシステムを作った. 一方で仲道は将棋においてライバル AI を実践しようとした [4]. 仲道の提案した手法は, 強さと着手の不自然さに基づいてのものである. 初心者の人間との対戦でも勝率を調整できるのか, 人間から見て明らかな悪手を指していないかの 2 点の検証をしライバル AI の調整をしようというものである.

1.4　本報告書の構成

本報告書の構成は以下の通りである. まず第 2 章にて, 本研究で作成したライバル AI とライバル AI の性能の検査に用いる 2 つの AI の仕様について述べ, 第 4 章で作成したライバル AI の性能を検査するためにライ

バル AI とライバル AI 自身を含めた 3 つの AI が対戦した結果を述べる. そして第 5 章では, 第 6 章の結果に対する考察を述べる.

2　研究内容

本章では本研究にて作成した将棋の AI プログラムについて述べる. 本研究では将棋のライバル AI（以下 RvlAI とする）を作成する. 本研究で作成する RvlAI は, 対戦相手の指した手の評価値の高さによって, 自身の指す手を変える AI である.RvlAI は, 盤面の評価値の値によってで自分の読みの深さを変えることで, 対戦相手の指した手が, 評価値の高い手には評価値の高い手, 評価値の低い手なら低い手を選択することにより, 相手と同程度の強さになるようにする. また,RvlAI は候補手の先読みの方法として $\alpha\,\beta$ 法を用いている. 盤面の評価方法と $\alpha\,\beta$ 法については以下の 2.1 節と 2.2 節で述べる. 本研究で作成されたプログラムのソースコードを付録に添付する.

2.1　評価方法

RvlAI には, 対戦相手と自分の実力を同じようにするために, 盤面評価を行う. 本節ではその評価方法を述べる. RvlAI では局面の評価方法として, 各駒に点数を割りその合計で評価する方法を用いている. 表 1 に各駒に割り振った点数を示す. 各駒の点数にその駒を持っているプレイヤーが先手なら +1, 後手ならば-1 をかけ, その合計値がプラスなら先手有利,, マイナスなら後手有利と評価する. 例えば図 3 に示す局面の場合, 後手が歩一個取られ先手の持ち駒になっているため, 駒の点数を合計すると,+200 となり先手有利となる.

表 1　各駒の点数

	歩	香	桂	銀	金	角	飛	玉
生駒	100	600	700	1000	1200	1800	2000	999999
成駒	1200	1200	1200	1200	1200	2000	2200	999999
持駒	100	600	700	1000	1200	1800	2000	無し

2.2　ミニマックス法

本節ではミニマックス法について述べる. ミニマックス法は探索木の葉から根に向かって評価値を求めていく手法である. ミニマックス法を用いて着手を選択する場合, まず各候補手に対して探索木を構成する. 各探索木の頂点が各局面を表し, 各頂点の子は 1 手先の局面を表す. 葉ではその時点での盤面の評価値を求める. 各頂点では先手なら最大の評価値を持つ子の値, 後手なら最小の評価値を持つ子の値をその頂点の評価値とする. この操作を全ての葉から根に向かって行い, 根の評価値を候補手の評価値とする. ミニマックス法は, 最終的にもっとも評価値の高い手を選択することができる手法である. しかしこの方法には欠点があり, それが探索時間によるものである.MinMax アルゴリズムでは,1 手先を読むごとに, その手番のプレイヤーの可能な手を全て読む必要がある. 実際にこのプログラムを将棋に使うと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, そのため, 序盤では 4 手の深さまで読んでも高速に動いていても, 終盤になった途端にとんでもなく遅くなってしまう.

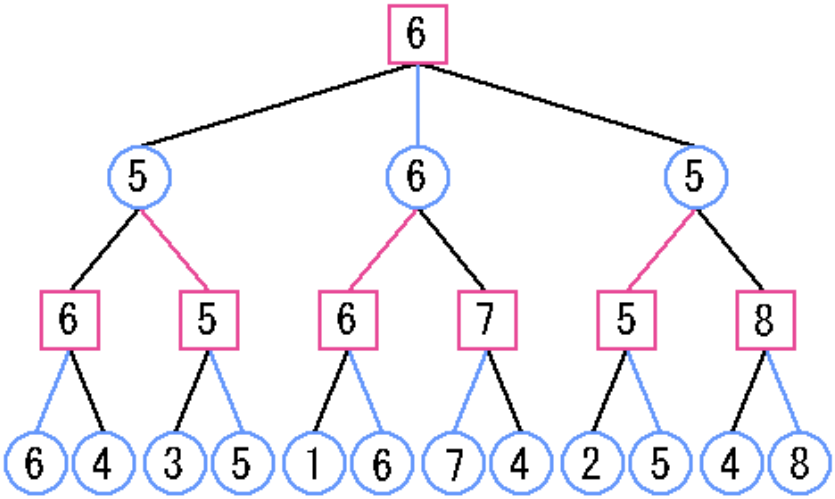

図 2　ミニマックス法の例

2.3　αβ法

RvlAI には, 候補手の評価値を先読みするためにαβ法が用いられる. αβ法とは探索アルゴリズムの一つでミニマックス法と同じ結果が得られるにもかかわらず, 理論上の計算量は, 同じ時間でほぼ 2 倍の深さまで読めるというアルゴリズムである. 探索する必要がない枝を探索しないための工夫である. αβ法による探索の省略の例を, 図 3 に示す. 三列目の F と G に注目する. 右に MAX と書かれているため,5 と 8 では 5 をとる. よって 8 である G に連なる手は全て考慮しない. そのため全ての手を考慮するミニマックス法よりも高速になる. またこのαβ法のミソであるαカットβカットについて説明する. その様子については図を見て欲しい.D

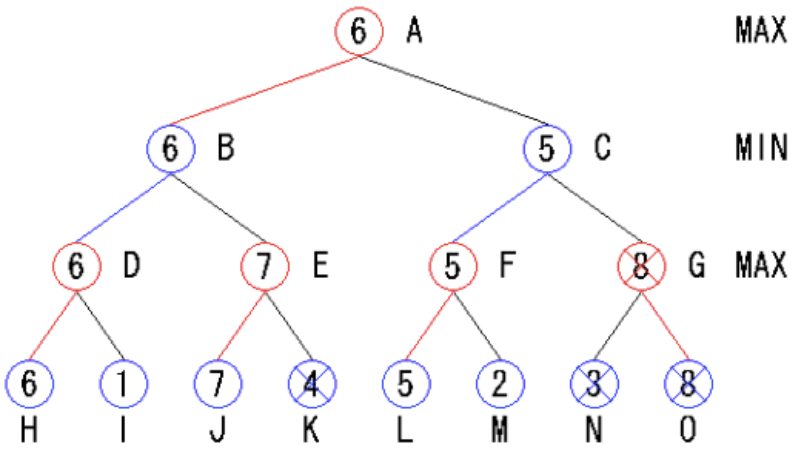

図 3　αβ法の例

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わかるため, それ以降を探索しない. しかしこの $\alpha\beta$ 法で最高速度を得るには, 探索の順番が完全に良い評価を返す順にソートされている必要がある.

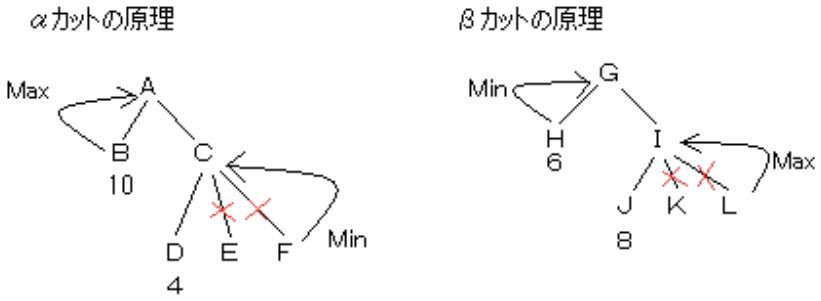

図 4　$\alpha\beta$ 法の例

3　着手の選択

本章では, 着手の選択方法について述べる. 本研究で作成するライバル AI は候補手の中から評価値を 0 に最も近づける手を指すタイプ A, 評価値によって読みの深さを変更するタイプ B, 序盤のみ定跡通りの手を指し中盤戦からはタイプ B の評価方法にもどすタイプ C のの 3 種の着手選択方法を用いる. タイプ A に関しては, 松村の研究でオセロに用いていたライバル AI のシステムを将棋に移植した物になる. タイプ B は参考文献 [4] から読みの深さによって評価値を設定していたのでそれをもとにライバル AI の読みの深さを変更すれば相手の棋力に合わせられるのではと考えた. タイプ C では序盤は定跡通りに動けるが中盤以降からは自分で考えるため弱くなってしまうのではと考えたためタイプ C で検証することにした.

4　ライバル AI プログラム

本章では, 本研究で作成した, ライバル AI プログラムについて説明する. 付録に本研究で作成したプログラムのソースを示す. 今回作成した RvlAI クラス, 定石データを用いたする方法に使った Joseki クラス, について説明する. しかし, このプログラムでは, 盤面の全部の駒, 全部の持ち駒について毎回計算を行っていた. これを,move() 関数で, 取った分の駒のぶんを点数に加算したり, 駒がなった際に点数を加算するような処理を入れ,back() 関数で, 取った駒の分を点数から減算するような処理を入れ, 評価関数（evaluate()）では, 単に「Kyokumen」で現在保持している値を返すようにしたほうが, 高速になるため実装している.

4.1　Kyokumen クラス

Kyokumen クラスは将棋の盤面を表すクラスである.Kyokumen クラスはフィールド変数として盤面を表す int 型の配列の ban[][] をもち, 局面の持ち駒の比較, 駒の種類ごとに枚数が同じかどうかの比較, 局面が同一かどうかの判断, 比較用の配列を準備などをこのクラスで行う. また, 今回の局面の評価を行うのもこのクラスで行う.

4.2　KyokumenKomagumi クラス

このクラスは Kyokumen クラスの継承で, 序盤戦, 中盤戦, 終盤戦における評価方法の追加ぶんを実装している. 序盤戦では玉を固める, 攻撃の態勢を作る, の二つ行動をバランスよく行う必要がある. また, 中盤戦, 終盤戦ではお互いの歩以外の持ち駒, お互いの相手陣 4 段目以内に侵入した駒に評価の基準のおもきをおいた. 序盤戦においては, 各駒が何段目にいるか, での点数ボーナス. 各戦型別に, 自分の駒に与える, 位置によるボーナス点. これは矢倉や美濃囲いなどの戦型に基づいている. 次に中盤戦, 終盤戦においても, 上の 2 項目から終盤度の計算（現在の局面が終盤戦または中盤戦に移っているかどうかの確認のため）, 終盤度によるボーナスの付加, 駒の加点などの点数の入力もここで行った.

4.3　RvlAI クラス

RvlAI クラスは対戦相手の打った手によって実力を変えるライバル AI を表すクラスである.RvlAI クラスのフィールド変数には読みの深さを表す DEPTH, 最善手順を格納する best[][] などがある. 定石を利用してす

る場合は Joseki クラスを利用するためにそのための変数 joseki も生成しておく.

4.4 negaMax メソッド

negaMax メソッドでは,2.3 節で述べた $\alpha\beta$ 法をを用いて候補手の評価値を返すメソッドである. 探索する深さは, 現在の盤面の評価値によって変更する. 最初の盤面は 0 なので DEPTH は 4 からスタートする. その深さで最善手を見つけた後その手を指した盤面の評価値で次の手を指す時の探索する深さを変更する. また, 序盤戦に定石を利用する場合は Joseki クラスを利用する.

5　着手選択とその実験的評価

本章では本研究で作成した RvlAI の実験的評価について述べる.

5.1　RvlAI の着手選択方法

RvlAI は松村の論文 [2] に従い盤面の評価値が 0 に最も近づける手を探す,RvlAI は先読みの盤面評価に基づいて選択基準とする.

5.2　RvlAI の実験的評価

本研究で作成した RvlAI の性能を実験的に評価するために,AI どうしで対戦を行った. 対戦相手として読みの深さが 2 の弱 AI と, 読みの深さが 4 の強 AI を生成した. 表 2 に AI どうしで 100 回ずつ対戦させた結果を示す. 表 2 に示される通り引き分けの数が多くなっている. 従って, ただ長く引き伸ばすだけの AI になってしまっている.

表 2　AI 同士の対戦結果（試行回数 100 回）

先手	後手	先手勝	引分	後手勝
RvlAI	強 AI	12	54	34
強 AI	RvlAI	47	45	8
RvlAI	弱 AI	40	43	17
弱 AI	RvlAI	3	64	33
RvlAI	RvlAI	43	0	57

そこで評価値の値によって読みの深さを変えるプログラムを作成した. 評価値の範囲については [4] を参考にした. その結果を表 3 に示す. 表 3 に示されるように引き分けが減少したのがわかる. また, 勝敗の内容についてだが強 AI との対戦については棋力合わせられているといえる. しかし弱 AI に関しては全く棋力を合わせることができなかった合わせることができなかった.

表 3　AI 同士の対戦結果（試行回数 100 回）

先手	後手	先手勝	引分	後手勝
RvlAI	強 AI	37	7	56
強 AI	RvlAI	60	5	35
RvlAI	弱 AI	90	6	4
弱 AI	RvlAI	1	4	95
RvlAI	RvlAI	54	0	46

次に序盤のみ飛車が中央から左に行くと美濃囲いの定跡の手を指しその後, 先ほどの RvlAI のプログラムにつなげる手法を行った. 今回は美濃囲いという定跡を採用する. その結果を表 4 に示す. 表 4 から少しだが 5 割

に近づいていることが分かる. また悪化している部分はランダム変数によるものだと考える. 今回は美濃囲いのみ実装しているが, これを将棋のことを勉強し, 定跡にはいる予備動作を知り実装すればさらに結果が向上すると思われる.

表 4　AI 同士の対戦結果（試行回数 100 回）

先手	後手	先手勝	引分	後手勝
RvlAI	強 AI	12	27	61
強 AI	RvlAI	58	6	36
RvlAI	弱 AI	85	4	11
弱 AI	RvlAI	5	2	93
RvlAI	RvlAI	38	0	62

6　結論・今後の課題

本研究では, 相手の実力と同じような強さで戦う将棋のライバル AI を作成した. 本研究で作成した RvlAI は, 強 AI, 弱 AI に対戦した結果, 目標である全ての対戦相手に対して, 勝率 5 割程度にすることはできなかった. よってプログラムに改善する必要があると考えられる. 改善点として, 評価基準をより人間の思考に寄せるために, 人と対戦することでそこから学習させる機械学習の実装や, 遺伝的アルゴリズムの採用をすることで対戦相手によってより多彩な戦略を持たせることによる勝率の安定などが挙げられる. また昨今注目を集めているディープラーニングを取り入れ強さの上限をあげることでさらに多くのプレイヤーに棋力を合わせられることにつながると考えられる. また, 今回は美濃囲いのみ序盤の敵の動きを実装したが, これを全ての定跡で実装すれば, 序盤戦は定跡通りの良い手を指すが, 中盤戦以降弱くなるという中級者に対しても自然に弱くなれると考えた.

謝辞

卒業研究においてだけでなく, 就職活動に渡ってサポートしていただき, 誠に感謝しております. これからの社会人生活にも活かせる経験をさせていただいたと感じております. また, 協力していただいた皆様に心から感謝の気持ちと御礼を申し上げます

参考文献

[1] 池　泰弘:Java 将棋のアルゴリズム, 工学社 (2007).

[2] 松村　憲樹:オセロにおけるライバル AI の作成について. 情報学科 2016 年度卒業研究報告書 (2017).

[3] 上田　陽平:遺伝的にアルゴリズムによる人間のレベルに対応する多様なオセロ AI の生成. 研究報告 ゲーム情報 学, Vol.2012-GI-27, No.5, pp.1-8, 情報処理学会 (2012).

[4] 仲道　隆史, 伊藤　毅志:プレイヤの技能に動的に合わせるシステムの提案と評価. 情報処理学会論文誌, Vol.57, No.11, pp.2426-2435, 情報処理学会 (2016).

[5] 斎藤 康毅　ゼロから作る Deep Learning —Python で学ぶディープラーニングの理論と実装, オライリージャパン (2016)

[6] 李 咏謙,Grimbergen, R.:評価特徴によるプレイヤーレベルに合わせるゲーム AI, ゲームプログラミングワークショップ 2012,pp.134136, 情報処理学会 (2012).

[7] 史上最強棋士はだれか 将棋 AI が出した答えは — DG Lab Haus https://media.dglab.com/2017/10/25-shogiai-01/

[8] 竹内 聖悟, 金子 知適, 山口 和紀, 将棋における, 評価関数を用いたモンテカルロ木探索ゲームプログラミングワークショップ 2010 論文集, Vol.2010, No.12, pp.86-89, 情報処理学会, (2010), http://id.nii.ac.jp/1001/00071322/

[9] 金子 知適 , 田中 哲朗 , 山口 和紀 , 川合 慧駒の関係を利用した将棋の評価関数の学習, 情報処理学会論文誌, Vol.48, No.11, pp.3438-3445 (2007), http://id.nii.ac.jp/1001/00009785/

付録 A　ソースプログラム

Listing 1　RivalAI.java

```
1  package lesserpyon;
2  import java.util.Vector;
3  import java.util.Random;
4
5  // コンピュータの思考ルーチン
6  public class RivalAI implements Player,Constants {
7              Random random ;
8              Random rnd = new Random();
9    // ∞を表すための定数
10   static final int INFINITE=99999999;
11
12   // 読みの深さ
13   static final int DEPTH_4=4;
14   static final int DEPTH_3=3;
15   static final int DEPTH_2=2;
16   static final int DEPTH_1=1;
17   // 読みの最大深さ…これ以上の読みは絶対に不可能。
18   static final int LIMIT_DEPTH=16;
19
20   // αβカットを起こす関係で、ランダム着手は出来なくなる。
21   // 詳細は解説にて。
22
23   // 最善手順を格納する配列
24   public Te best[][]=new Te[LIMIT_DEPTH][LIMIT_DEPTH];
25   public Te best1[][]=new Te[LIMIT_DEPTH][LIMIT_DEPTH];
26
27   // この思考用の TransPositionTable
28   TranspositionTable tt=new TranspositionTable();
29
30   int leaf=0;
31   int node=0;
32
33   // 定跡があれば定跡を利用。
34   Joseki joseki;
35
36   public RivalAI() {
37     joseki=new Joseki("public.bin");
38   }
39   //タイプの場合 Ak,で返ってくる値を絶対値にして 0 に最も近い手を返す evaluate
40   //ただし最善手順には影響させない
41   int negaMax(Te t,Kyokumen k,int alpha,int beta,int depth,int depthMax) {
42     // 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。
43     if (depth>=depthMax) {
44       leaf++;
45       if (k.teban==SENTE) {
46         return k.evaluate();
47       } else {
48         return −k.evaluate();
49       }
50     }
51     node++;
52     TTEntry e=tt.get(k.HashVal);
```

```
53      if (e!=null) {
54        if (e.value>=beta && e.depth<=depth && e.remainDepth>=depthMax−depth &&
55            e.flag!=TTEntry.UPPER_BOUND) {
56          return e.value;
57        }
58        if (e.value<=alpha && e.depth<=depth && e.remainDepth>=depthMax−depth &&
59            e.flag!=TTEntry.LOWER_BOUND) {
60          return e.value;
61        }
62      }
63      // 現在の指し手の候補手の評価値を入れる。
64      int value=−INFINITE;
65
66      // 最初に軽い手の生成をしてみる。
67      Vector v=GenerateMoves.makeMoveFirst2(k,depth,this,e);
68
69      for(int i=0;i<v.size();i++) {
70        // 合法手を取り出す。
71        Te te=(Te)v.elementAt(i);
72
73        // その手で一手進める。
74        k.move(te);
75        // では、先手後手を入れ替えないので…。move
76        if (k.teban==SENTE) {
77          k.teban=GOTE;
78        } else {
79          k.teban=SENTE;
80        }
81
82        // その局面の評価値を、さらに先読みして得る。
83        Te tmpTe=new Te(0,0,0,false,0);
84        int eval=−negaMax(tmpTe,k,−beta,−alpha,depth+1,depthMax);
85        k.back(te);
86        // では、先手後手を入れ替えないので…。back
87        if (k.teban==SENTE) {
88          k.teban=GOTE;
89        } else {
90          k.teban=SENTE;
91        }
92
93        // 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？
94        if (eval>value) {
95          // 返す値を更新
96          value=eval;
97          // α値も更新
98          if (eval>alpha) {
99            alpha=eval;
100         }
101         // 最善手を更新
102         best[depth][depth]=te;
103         t.koma =te.koma;
104         t.from =te.from;
105         t.to =te.to;
106         t.promote=te.promote;
107         // 最善手順を更新
108         for(int j=depth+1;j<depthMax;j++) {
109           best[depth][j]=best[depth+1][j];
```

```
110             }
111             // βカットの条件を満たしていたら、ループ終了。
112             if (eval>=beta) {
113               tt.add(k.HashVal,value,alpha,beta,best[depth][depth],
114                   depth,depthMax−depth,0);
115               return eval;
116             }
117           }
118         }
119
120         // 現在の局面での合法手を生成
121         v=GenerateMoves.generateLegalMoves(k);
122
123         GenerateMoves.evaluateTe(k,v);
124
125         // 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。
126         //弱くするのはここの部分を変える
127         for(int i=0;i<v.size();i++) {
128           // 合法手を取り出す。
129           Te te=(Te)v.elementAt(i);
130           if (te.value<−100 && value>−INFINITE) {
131             break;
132           }
133
134           // その手で一手進める。
135           k.move(te);
136           // では、先手後手を入れ替えないので…。move
137           if (k.teban==SENTE) {
138             k.teban=GOTE;
139           } else {
140             k.teban=SENTE;
141           }
142
143           // その局面の評価値を、さらに先読みして得る。
144           Te tmpTe=new Te(0,0,0,false,0);
145           int eval=−negaMax(tmpTe,k,−beta,−alpha,depth+1,depthMax);
146           k.back(te);
147           // では、先手後手を入れ替えないので…。back
148           if (k.teban==SENTE) {
149             k.teban=GOTE;
150           } else {
151             k.teban=SENTE;
152           }
153
154           // 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？
155           if (eval>value) {
156             // 返す値を更新
157             value=eval;
158             // α値も更新
159             if (eval>alpha) {
160               alpha=eval;
161             }
162             // 最善手を更新
163             best[depth][depth]=te;
164             t.koma =te.koma;
165             t.from =te.from;
166             t.to =te.to;
```

```
167             t.promote=te.promote;
168             // 最善手順を更新
169             for(int j=depth+1;j<depthMax;j++) {
170               best[depth][j]=best[depth+1][j];
171             }
172             // βカットの条件を満たしていたら、ループ終了。
173             if (eval>=beta) {
174               break;
175             }
176           }
177         }
178         tt.add(k.HashVal,value,alpha,beta,best[depth][depth],
179             depth,depthMax−depth,0);
180         return value;
181       }
182
183       int ITDeep(Kyokumen k,int alpha,int beta,int depth,int depthMax) {
184         int retval=−INFINITE;
185         int i;
186         Te te=new Te(0,0,0,false,0);
187         for(i=depth+1;i<=depthMax;i++) {
188           retval=negaMax(te,k,alpha,beta,depth,i);
189         }
190         return retval;
191       }
192
193       public Te getNextTe(Kyokumen k,int tesu,int spenttime,int limittime,int byoyomi) {
194         leaf=node=0;
195
196         Te te;
197         long time=System.currentTimeMillis();
198
199         if ((te=joseki.fromJoseki(k,tesu))!=null) {
200           System.out.println("定跡より:"+te.toString());
201           return te;
202         }
203
204         // 評価値最大の手を得る
205         // 投了にあたるような手で初期化。
206         KyokumenKomagumi kk=new KyokumenKomagumi(k);
207
208         te=new Te(0,0,0,false,0);
209         int x= 0;
210         //ここで盤面の評価値より棋力を変動
211         if(k.evaluate()<1500){
212               x = DEPTH_4;
213         }else if(1500<=k.evaluate()&&k.evaluate()>4000){
214               x = DEPTH_2;
215         }else{
216               x = DEPTH_1;
217         }
218         int v=ITDeep(kk,−INFINITE,INFINITE,0,x);
219         if (v>−INFINITE) {
220           te=best[0][0];
221         }
222         //盤面を表示する際にはここに書く
223         System.out.print("先読みの評価値:"+v);
```

```
224       System.out.print("␣␣最善手順:");
225       for(int i=0;i<x;i++) {
226         System.out.print(best[0][i]);
227       }
228
229       System.out.println();
230
231       time=System.currentTimeMillis()−time;
232       System.out.println("leaf="+leaf+"␣node="+node+"␣time="+time+"ms");
233       return te;
234     }
235 }
```