卒業研究報告書

題目

アンパンマン将棋の完全解析

指導教員

石水　隆　講師

報告者

08-1-037-0094

滝口　直

近畿大学理工学部情報学科

平成 24 年 1 月 31 日提出

概要

本研究では、セガトイズから 2012 年 6 月 28 日に幼児用として発売された「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)を題材とする。アンパンマン将棋の大きな特徴は、「３×５の盤」「タッチダウン制」「３種類６つの駒」「取った駒は使えない」「後ろ方向に移動できない」ということである。

本研究では、アンパンマン将棋の完全解析を最終目標とする。完全解析に先立ち、まずアンパンマン将棋の java アプリケーション開発を行う。本研究により作成したアプリケーションは対 AI 戦を行うことができアンパンマン将棋を一人で遊ぶ事が可能になった。

本研究の AI は着手可能手の集合の中から一番評価値の高いものを選びゲームを有利に進めていく。評価値の決定は盤上の駒の数とその位置、着手可能手数により行う。評価値の計算は、各各着手可能手各着手可能手に対して、その手を指した場合の次局面の生成を再帰的に行い、先読み手数が一定値に達した場合駒の配置からその局面の評価値を求め、着手可能手を指した場合の次局面の評価値のうち最高値をその局面の評価値とする。AI は、着手可能手のうち最も高い評価値が得られる局面を生成する手を採用する。

本研究で作成したＡＩ同士の対戦を 100 回行ったところ、先手の 33 勝 53 負 14 引き分けであった。このことから、後手は先手の手を見てから後手の一手目を指すことができるので有利にゲームを進められるので後手有利なのではないのかと推論する。しかし先手も最善手を尽くせば引き分けになるのでないのかと考える。

また、本研究では、限定された局面に対する部分解析を行った。双方の初手をリーダーが前に出るものに制限した場合、初手リーダーB4 で先手有利となることを示した。

目次

1　序論

1.1 本研究の背景

アンパンマンはじめてしょうぎ[1](以下アンパンマン将棋とする)は 2012 年 6 月 28 日に発売された女流棋士の北尾まどか初段によって考案された二人完全零和有限確定完全情報ゲームである。

将棋やチェス等に代表されるボードゲームは、二人零和有限確定完全情報ゲームに分類される。零和とは、勝ちを＋１、負け－１、引分けを０として、全プレイヤーの数値の合計が常に０（一定）となるゲームのことである。有限とは、各プレイヤーの可能な手の組み合わせ総数が有限であるゲームの場合である。将棋やアンパンマン将棋は「千日手」、チェスでは「パペチュアル(perpetual)」、囲碁では、相手の石を取ると即座に相手が取れる状態になることを「劫(こう)」といい盤面上に「劫」が３ヶ所できることを「３劫」という。この劫を順に打っていくと永久にゲームが続くことになる。両者が譲らない場合は無勝負になり打ち直しになる[18]。確定とは自分と相手の着手がわかれば、ゲームの経過が完全に決まる場合である。完全情報ゲームとは、局面の情報が両プレイヤーに全ての情報を公開しているゲームのことである。二人零和有限完全情報ゲームは、その性質上解析を行い易いため、ゲーム理論において様々な研究がなされてきた。また、人工知能の分野においても広く研究がなされている[2]。

1.2 二人零和有限確定完全情報ゲームの完全解析に関する既知の結果

二人零和有限確定完全情報ゲームは全ての局面が決定可能なので、双方最善手を指した場合、先手勝ち、後手勝ち、引き分けのどれになるかはゲーム開始時点で決定しており、理論上、全ての可能な局面を解析することができれば最善の手を打つことができる。しかし多くの二人完全零和有限確定完全情報ゲームでは、可能な局面の総数が極めて大きいため、完全解析を行うことは不可能である。例を挙げれば、可能な局面数はリバーシが 10^{28} 通り、チェスが 10^{50} 通り、将棋が 10^{69} 通り、囲碁が 10^{170} 通り程度あるとされており、現在の計算機の性能を越えている。一方、二人零和有限確定完全情報ゲームのうち可能な局面数が少ない五目並べ、三目並べ、チェッカー、6×6 のオセロは必勝法がすでに知られている。連珠は双方最善手を打った場合、47 手で先手が勝つ[6]。チェッカーは双方最善手を指すと引き分けとなる[7]。他にも、Hex は背理法を用いて先手必勝であり[19]、2005 年に発売した SIMPEI は後手必勝であり最長手数は 49 手であることが判明している[20]。

局面数が大きいゲームについては、ゲーム盤をより小さいサイズに限定した場合の解析も行われている。サイズ 6x6 のリバーシでは、双方最善手を打つと 16 対 20 で後手勝ちとなる[8]。また、サイズ 4x4 の囲碁は双方最善手を打つと持碁(引き分け)[9]、5x5 の囲碁は黒の 25 目勝ちとなる[10]。

将棋については、盤面のサイズや使用する駒の種類を減らしたサイズ５五将棋[11]やゴロゴロ将棋[12]などのミニ将棋がある。５五将棋はサイズ 5x5 の盤と 6 種類の駒、ゴロゴロ将棋はサイズ 5x6 の盤と 4 種類の駒を使用する。図 1，図 2 に５五将棋およびゴロゴロ将棋の盤と駒の初期配置を示す。これらは本将棋と比べて可能な局面数が少ない。しかしながら現在のところまだこれらは完全解析されていない。

完全解析されているミニ将棋として、同じ考案者で 2008 年に発売され、アンパンマン将棋発売のきっかけとなった「どうぶつしょうぎ」[13](以下動物将棋とする)がある。動物将棋はサイズ 3*4 の盤と、ライオン，象，キリン，ひよこの 4 種類の駒を使用する幼児向けのミニ将棋である。図 3 に動物将棋の盤と駒の初期配置を示す。

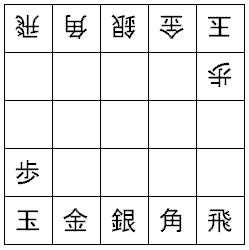


図１ ５五将棋の盤面と駒の初期配置

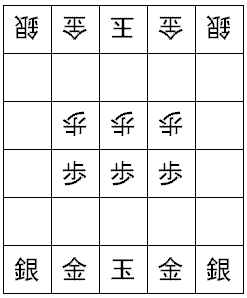
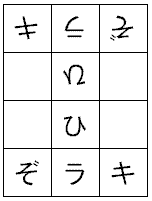


図２ ゴロゴロ将棋の盤と駒の初期配置



ラ：ライオン
ぞ：象
キ：キリン
ひ：ひよこ

図３ どうぶつしょうぎの盤と駒の初期配置

動物将棋は後退解析（retrograde analysis)により双方最善手を指した場合、初期局面が 78 手で後手の勝ち局面となり後手必勝ということが判明している[5]。

後手解析の手順は、勝負がついた局面から一つ手前の局面に戻る。負け局面から１手前の局面は勝ち局面であり、勝ち局面から１手前の局面が未確定の場合、そこから可能な手が全ての勝ち局面が勝ち局面になる場合は負け局面とする。この手順で初期配置が勝ち局面か負け局面かを導き出すことができる。本将棋など局面数が膨大な二人完全零和有限確定完全情報ゲームでは不可能だが動物将棋やアンパンマン将棋などある程度局面数が限られている場合では有効な解析手法である。

1.3 完全解析されていない二人零和有限確定完全情報ゲームに対する手法

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確実な最適手を得ることはできないが、局面の評価値計算、定跡データベース、一定手数の先読み、終盤での必勝読みと完全読み、モンテカルロ法などを用いてより有利だと思われる手を選択することができる。

局面の評価値計算とは、現在の局面が有利か不利なのかを数値にして比較するための計算である。評価値の設定は重要であり、もしも評価値設定が間違えるようならプログラムは自ら不利になるような手を指してしまう。将棋ならば、駒得か駒損か、王の囲いなどを複合的に計算する必要がある。

定跡とは、序盤での決まった指し手の形である。将棋の歴史が始まったときから研究がなされており、序盤戦は優劣がつけにくい場合が多いので定跡通りに序盤を進めることができるかどうかは非常に大きな問題となっている。定跡をデータベースに保持しおくことにより、定跡の返しが行える。また、データ通りに指すので相手が定跡通りに追従してくれる場合には、考えることなく指すことできる。

一定手数の先読みとは、ゲーム木を適当な深さまで調べてそこで局面の静的評価を行う。再帰的手続きでプログラムを書き、順に評価値を計算していく。先読みの基本原理は相手が常に自分の考える最善手を指してくるという行動前提の min-max 戦略である。

終盤での必勝読みと完全読みとは、終盤では総局面数が限られているので、コンピュータの計算により最

終手番まで全着手可能手を導き出し完全読みきることである。詰将棋の創作ではコンピュータを使用して作品の不詰や、余詰がないかを確認をする。

モンテカルロ法とは、シミュレーションなどにより乱数を用いて実行する手法の総称である。円周率の計算でよく用いられる。1×1 の正方形の中に 0.5 の円を描き、ランダムで点を打っていく。このとき「正方形内部の点の数：円中点内部の点の数＝正方形の面積：円の面積」となる。これを用いることで円の面積が算出され、円の面積の公式と正方形の面積のから円周率を計算することができる。ボードゲームへ応用するならば、ある局面からランダムに手を指しつつけたときの勝率を計算して勝率が高い局面ほど優れた局面と判断する。しかし、完全なランダムではいい結果を得られることは少なく、ゲームの知識を利用して指しての絞り込みや確率の選択を行うことが一般的であるといえる。オセロでは一定数内に終局するが、将棋では終局まで予測が難しく数も多いので予測率を保ちつつ、終局率の改善する必要がある[21]。

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

チェスでは、1997 年 5 月にチェスプログラム Deep Blue[14]が世界チャンピオン Garry Kimovich Kasparov と対戦を行い 2 勝 1 敗 3 引き分けで勝った[15]。将棋では、将棋プログラムボンクラーズ[16]が 2012 年 1 月に元プロ棋士の米長邦雄永世棋聖と対戦しボンクラーズ先手 113 手で勝った[17]。オセロでは、オセロプログラムの「ロジステロ」[23]と元日本チャンピオン富永健太氏の 2 番勝負が行われ、ロジステロ先手 38 対 26 でロジステロの 12 目勝ち、富永氏先手 23 対 41 でロジステロの 18 目勝ち、とロジステロが 2 勝した[22]。

1.4 本研究の目的

前述の通り、動物将棋は完全解析されており、双方最善手を打つと後手勝ちとなることが判明している。一方、アンパンマン将棋は発売が新しく未だ完全解析されていない。理論上動物将棋と同じくアンパンマン将棋も初期局面が必勝局面なのか引き分け局面なのかを探ることができると予想される。そこで本研究では、アンパンマン将棋の完全解析を目指す。アンパンマン将棋の解を出すことは、ゲーム研究の最終目標であり、まだ完全解析が行われていないゲームの向上につながる。また、アンパンマン将棋では千日手が非常に多く発生する事から局面のループが発生し易い本将棋や倉庫番ゲームなどの解を得る探索にループを多く持つ木の探索向上に役立つと考えられる。

1.5 本報告書の構成

本報告書の構成は、以下の通りである。

第 2 章は 2 つの節に分かれている。第 2.1 節では、本研究を行なっていく上で、アンパンマン将棋の細かなルールを説明する。第 2.2 節では、アンパンマン将棋のゲームとしての複雑さの指標を示す。

第 3 章では、本研究内容の説明をする。第 3.1 節では、本研究で用いた手法について、第 3.2 節では、アンパンマン将棋の部分解析について説明する。また、第 3.3 節ではアンパンマン将棋プログラムについて述べる。

第 4 章では本研究内容から検討を行い、考察を行う。

第 5 章は、第 4 章から得た研究内容の結論を述べ今後の課題を検討する。

なお、巻末には本研究で作成したプログラムと参考資料を添えた。

2　アンパンマン将棋

本節では、アンパンマン将棋について説明する。アンパンマン将棋の大きな特徴は、「３×５の盤」「タッチダウン制」「３種類６つの駒」「取った駒は使えない」「後ろ方向に移動できない」ということである。将棋よりもチェスに近く、動物将棋版のチェスと言える。以下にアンパマン将棋のルール説明を記述する。

2.1 アンパンマン将棋の将棋盤と駒

アンパンマン将棋は、サイズ 3×5 の将棋盤とアンパンマン，食パンマン，カレーパンマン，バイキンマン，ホラーマン，ドキンちゃんの 6 個の駒を使う。図 1 にアンパンマン将棋の将棋盤と駒の初期配置を示す。5 段目(A5, B5, C5)を「アンパンマンチーム」の陣地と呼び、「バイキンマンチーム」のゴールとなる。同じく、1 段目(A1, B1, C1)が「バイキンマンチーム」の陣地であり、「アンパンマンチーム」のゴールとなる。

アンパンマン将棋では、次の 6 個の駒を使用する。アンパンマンとバイキンマン(以下リーダーとする)は前や横、前斜め方向の５方向のいずれかに１マス進むことができる。本将棋では玉将に相当する駒で、アンパンマン将棋では「リーダー」と呼ぶ。カレーパンマンとドキンちゃんは、前と前斜めの３方向のいずれかに１マス進むことができる。しょくぱんまんとホラーマンは、前と横の３方向のいずれかに１マス進むことができる。

2.2 アンパンマン将棋の進行と勝敗

アンパンマン将棋は、将棋と同様にプレイヤー2 人で遊ぶゲームであり、図１の初期局面から交互に１手ずつ駒を動かす。駒は種類ごとに定められた動ける方向に１マス移動し、移動する先に敵の駒がある場合には捕まえる事ができる。アンパンマン将棋の駒はチェスと同じく取り捨てであり、捕まえた駒はゲーム終了まで盤の中に戻ることはできない。

アンパンマン将棋では、相手のリーダーを先に捕まえる、あるいは、自分のリーダーが先にゴールできたら勝ちとなる。リーダーを捕まえるとは、将棋で言えば相手の玉将を詰みにすることである。すなわち、相手のリーダーに対して王手が掛かっている状態で、相手の手番でその王手を回避できる手が無い場合に詰みとなる。相手の陣地にリーダーが入るとは、リーダーが最前列のマスに進むことである。つまり、アンパンマンは A1, B1, C1 のいずれかに、バイキンマンは A5, B5, C5 のいずれかに進むとゴールとなる。

また、駒の配置が同じ局面に３回目に到達すると引き分けになる。これを千日手と呼ぶ。連続して王手かけ千日手に判定されると王手をかけている側は負けとなる

2.3 アンパンマン将棋の総局面数

本節ではアンパンマン将棋で可能な局面数について考える。まずアンパンマンは盤内のいずれかのマスに配置されるので、15 通り、バイキンマンは盤外とアンパンマンのある位置、アンパンマンの隣・前・斜め前には置けないので 11 通り、食パンマンとホラーマンは盤外を含めどこでも置けるので各 16 通り、カレーパンマンとドキンちゃんは初期位置から到達できないマスが 3 箇所あるので各 13 通り、総局面を見積もると 7,138,560 通りである。この局面数は、完全解析されている動物将棋の可能な局面数の 1,567,925,964 通り[5]と比べても充分に小さい。よって、アンパンマン将棋の完全解析を行うことは充分に可能である。

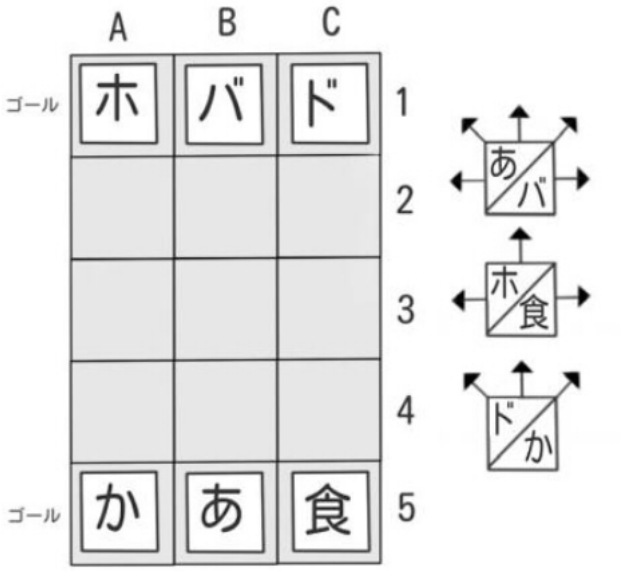


図 4：アンパンマン将棋の初期配置

3　研究内容

本研究では、アンパンマン将棋の完全解析に先立ちアンパンマン将棋の対人、対 AI 戦を行うことのできるプログラムを作成した。対 AI については各着手可能手から得られる局面を先読みし、それを元に各手の評価値を求め、適切な評価値が最大となる手を指していく AI を採用した。

3.1 アンパンマン将棋 AI で用いた手

1.3 節で述べたように、次に指すべき手をどのように選択するかは様々な手法がある。本節ではアンパンマン AI で用いた手法について説明する。

3.1.1 着手可能手の発見

ある局面で指すことができる着手可能手は、各自駒が各マスへ移動可能か判定を行うことによって求まる。ただし、リーダーの自殺手を避けるためリーダーは相手の駒が効いているマスには移動できないとする。また、王手を打たれた時は、王手から抜けられない手は除外し、王手から抜けられる手のみを有効手とする。有効手がない場合は詰み状態となり負けとなる。

3.1.2 局面の評価値計算

ある局面の評価値は、盤上に存在する駒の種類やその位置から決定される。一般に将棋は自駒が多いと有利、相手駒が多いと不利であるので、自駒に正の価値、相手駒に負の価値を付け、その合計値を評価値の基準のひとつとして用いる。また、リーダーは最前列まで進むと勝ちが決まるので、リーダーは前進に従い評価値を高く設定する。また、一般的にある局面で指せる手が多いほど有利であると考えられるため、着手可能手の数も評価基準としている。ただし、すでに勝負がついた局面は勝利局面なら評価値を無限大に、負なら評価値無限

小に、引き分けなら０にする。評価値の計算は、各着手可能手に対して、その手を指した場合の次局面の生成を再帰的に行い、先読み手数が一定値に達した場合駒の配置からその局面の評価値を求め、着手可能手を指した場合の次局面の評価値のうち最高値をその局面の評価値とする。AI は、着手可能手のうち最も高い評価値が得られる局面を生成する手を採用する。

3.1.3 王手の判定

王手の判定は、自分のリーダーの座標が相手リーダー以外の駒が次の手番で動できる座標に含まれているかをチェックする。例えば、ホラーマンの前と両横のマス、ドキンちゃんの前方 3 マスにアンパンマンがいるのかを調べる。

3.1.4 勝敗の判定

相手のリーダーを詰めるか、自分のリーダーが相手ゴールにいる場合勝利したと見なす。相手リーダーが詰んでいるかどうかは、3.1.1 節で述べた有効手があるかどうかと、3.1.2 節で述べた王手が掛かっているかどうかを判定すればわかる。

3.1.5 千日手の判定

千日手判定は、盤上の駒の位置を覚えておき、前から順番に比較して同じ局面の配列が 3 回目現れたらそこで千日手とする。

3.2 アンパンマン将棋の部分解析

アンパンマン将棋の完全解析は全ての局面の組み合わせを網羅する必要がある。一方、特定の局面については、比較的容易に解析できる場合もある。そこで、本研究では、いくつかの限定された局面について解析を行う。

3.3 アンパンマン将棋プログラム

本節では、アンパンマン将棋プログラムについて述べる。付録 1 に、アンパンマン将棋のソースを示す。

3.3.1 クラス Anpanman

Anpanman クラスは、実行クラスである。実行時、将譜を anpanmanScore.txt に出力する。ゲームの手続きはすべて while 文でくくりゲームのターンを再現する。

3.3.2 クラス Board

Board クラスはゲームのための手続きを行うクラスである。board 配列で盤外含め 5× 7 で表現している。プレイヤーの手番の処理を行う player メソッドは入力された指示が正しいのかを調べ、movePiece メソッドは、

将譜を作成後移動先に駒が場合は取り除き駒を移動させる。removePiece メソッドは駒を取り除くメソッドである

3.3.3 クラス Piece

Piece クラスは、駒を管理するクラスである。駒の移動できる方向、初期の位置や指定した位置にセットする駒が指定した座標に移動できるか、移動可能なリストを返す、移動可能な座標を表示する、クローン生成などのメソッドが存在する。

3.3.4 クラス NextMove

NextMove クラスは、駒の移動可能な位置を表すクラスである。移動する駒の種類、座標位置、評価値を返すメソッドや駒の種類、座標位置、評価をセットするメソッドがある。

3.4 アンパンマン将棋の計算機実験

本研究では、アンパンマン将棋で先手が有利なのか後手が有利なのかを推測するために、AI 同士で対戦させその勝率を計測する。本研究では先読み手数 5 手の AI 同士での対戦を 100 回行った。

4 結果および考察

4.1 部分解析結果の検討

本節では初期局面からの先手それぞれの合法手からの応手を考える。アンパンマンとバイキンマンが一列進んだ場合はコンピュータで解析しなくても有効な着手可能手の少なさから必勝法がわかる。

- 初手アンパンマン C4 の場合：バイキンマン B2 でバイキンマン必勝
- 初手アンパンマン B4 の場合：バイキンマン A2,B2 はアンパンマン必勝

他のアンパンマン、バイキンマンが一列進んだ場合は必勝法を確認したが、着手可能手が列挙するには多いので、やはりコンピュータ解析が必要となる。

4.1.1 1. ア B4, バ B2 の場合

1. ア B4, バ B2, の場合、2. カ A4 (図 5)と指すとバイキンマン側の手は、自殺手を除き バ A2, バ C2, ホ B1, ホ A2, ド C2 の 5 つの手が存在するが、全てバイキンマンの前にカレ―パンマンを指され負けとなる。図 5 で示す斜線マス A3,B3 に同時に利かせる駒が無いためである。すなわち、

2. カ A4, バ A2, 3. カ A3#
2. カ A4, バ C2, 3. カ B3#
2. カ A4, ホ B1, 3. カ B3#
2. カ A4, ホ A2, 3. カ B3#
2. カ A4, ド C2, 3. カ A3#

で 3 手で先手の勝ちとなる。

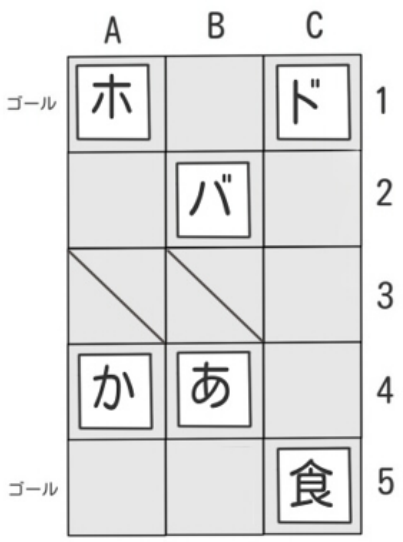


図 5: 1. ア A4, バ B2, 2. カ A4

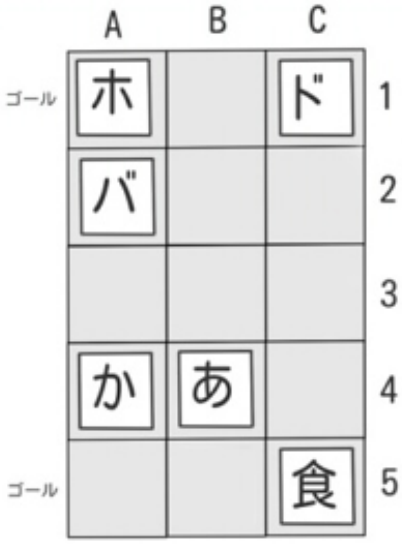


図 6: 1. ア A4, バ A2, 2. カ A4

4.1.2 1. ア B4, バ A2 の場合

1. ア B4, バ A2 の場合、先手が 2. カ A4 (図 6)と指すと、後手が指せる手は自殺手を除くと バ B2、ホ B1、ド B2, ド C2 の 4 つの手がある。バ B2、ホ B1、ド C2 は、

2. カ A4, バ B2, 3. カ B3

2. カ A4, ホ B1, 3. カ A3 (図 7)

2. カ A4, ド C2, 3. カ A3 (図 8)

で後手詰みとなる。よって、2...., ド B2 を指さなければならない(図 9)。ここで先手が 3. 食 C4 と指すと、後手バイキンマンは動けず、ドキンちゃんを動かす手は

3. 食 C4, ド A3+, 4. カ xA3# (図 10)

3. 食 C4, ド B3+, 4. カ B3# (図 11)

と後手詰みになるか、

3. 食 C4, ド C3+, 4. ア xC3 (図 12)

で 2 手後にアンパンマンのゴールが確定する。よって 3. ホ B1 のみである。

4 手目、先手はカレーパンマン、後手はドキンちゃんは動かすと負けが確定するので先手は食パンマン、後手はホラーマンのみを動かせる。よって 4 手目まででは、

1. ア B4, バ B2

2. カ A4, ド B2

3. 食 C4, ホ B1

4. 食 C3, ホ C1

と進み、図 13 の示す状態となる。ここで先手が 5. 食 B3 を指すと、ドキンちゃんを動かす手は、

5. 食 B3, ド A3, 6. カ xA3 (図 14)

5. 食 B3, ド B3, 6. カ xB3 (図 15)

5. 食 B3, ド C3, 6. 食 xC3 (図 16)

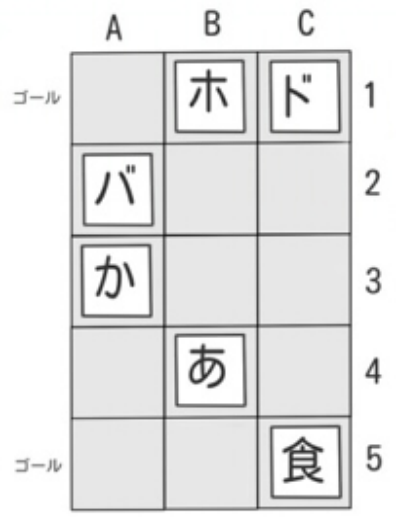


図 7:2. カ A4, ホ B2, 3. カ A3#

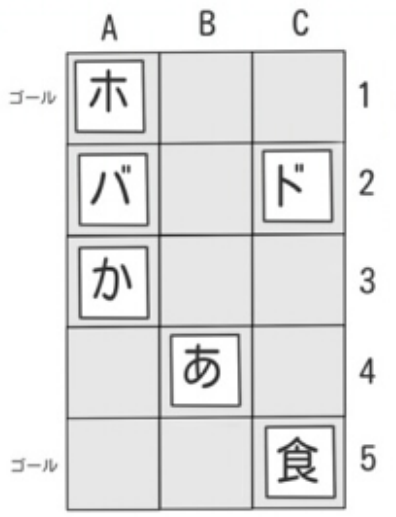


図 8:2. カ A4, ド C2, 3. カ A3#

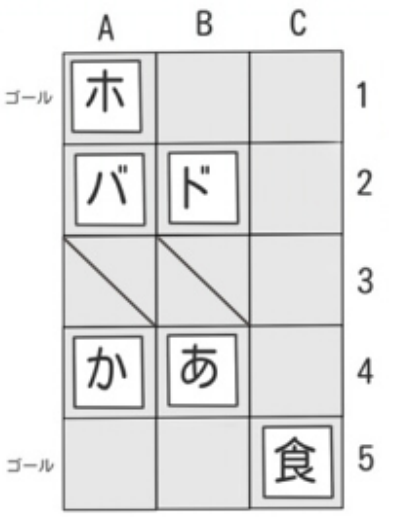


図 9：2. カ A4, ド B2

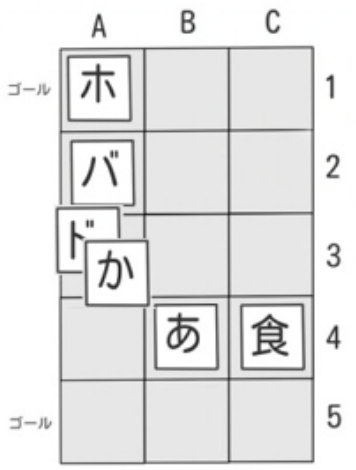


図 10:3. 食 C4, ド A3+, 4. カ xA3#

図 11:3. 食 C4, ド B3+, 4. カ xB3#

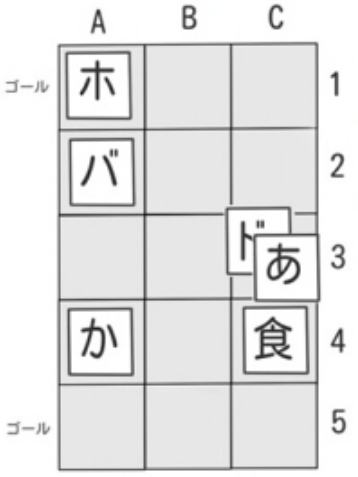


図 12:3. 食 C4, ド C3+, 4. ア xC3

で負けとなる。よってホラーマンを動かす 5. ホ B1 または 5. ホ C2 のみであるが、どちらの場合も、

6. 食 B2+, ホ xB2, 7. カ A3#, (図 17)

6. 食 B2+, バ xB2, 7. カ B3#, (図 18)

で詰みとなる。よって、1. ア B4, 2. バ A2 は 7 手で先手勝ちである。

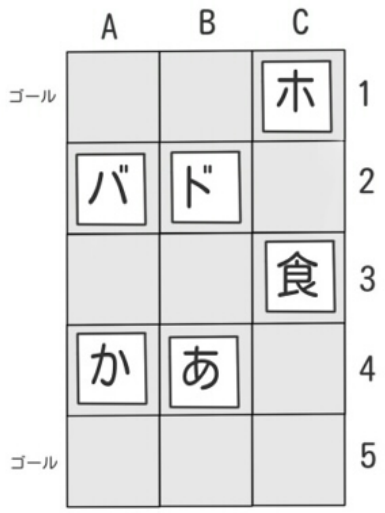


図 13:1. ア B4, バ B2, 2. カ A4, ド B2, 3. 食 C4, ホ B1, 4. 食 C3, ホ C1

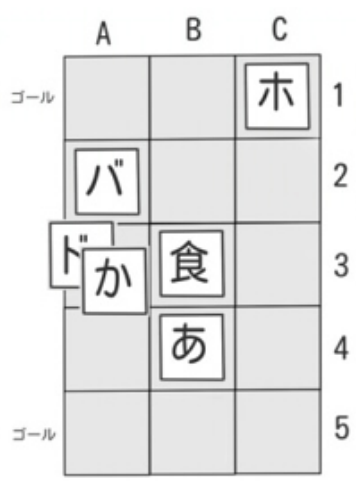


図 14:5. 食 B3, ド A3+, 6 カ xA3#

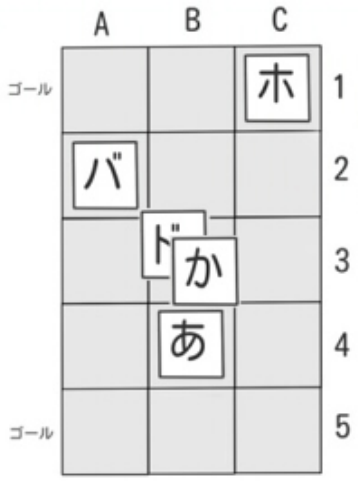


図 15:5. 食 B3, ド xB3+, カ xB3#

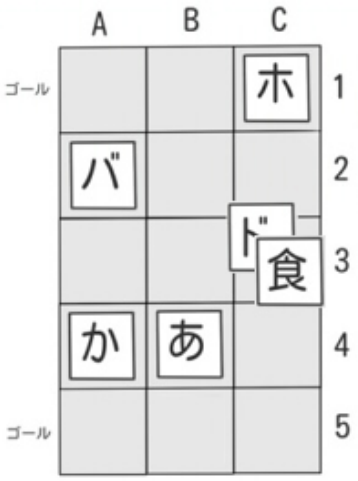


図 16:5. 食 B3, ド C3, 食 xC3

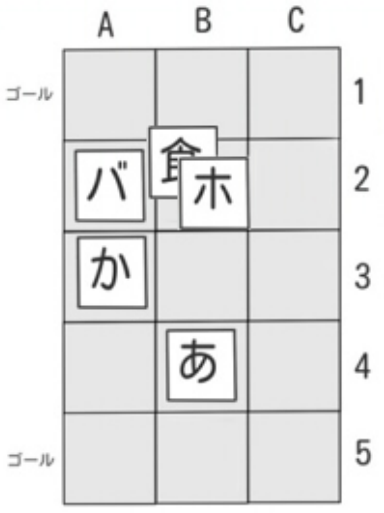


図 17:6.食 B2+, ホ xB2, 7 カ A3#

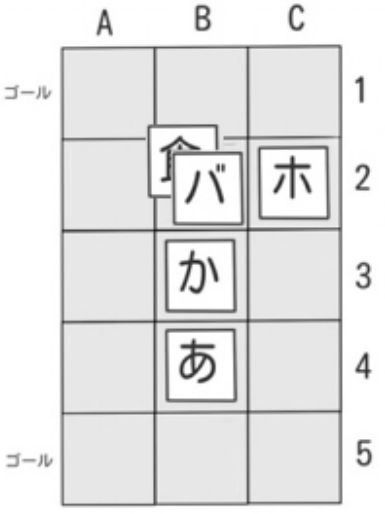


図 18:6.食 B2+, バ B2, 7. カ B3#

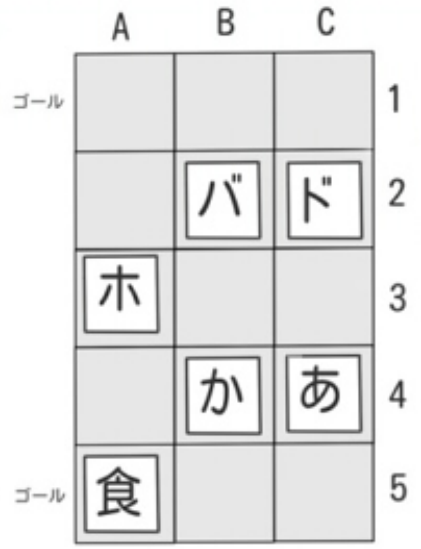


図 19:3. 食 B5, ホ A2, 4. 食 A5, ホ A3

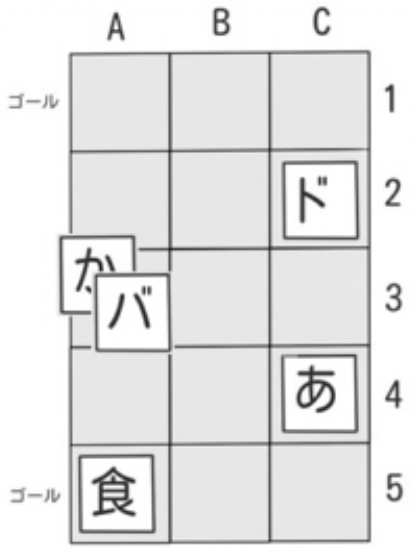


図 20 : 5. カ A3, バ A3

図 21:5.カ B3+, ド xB3#

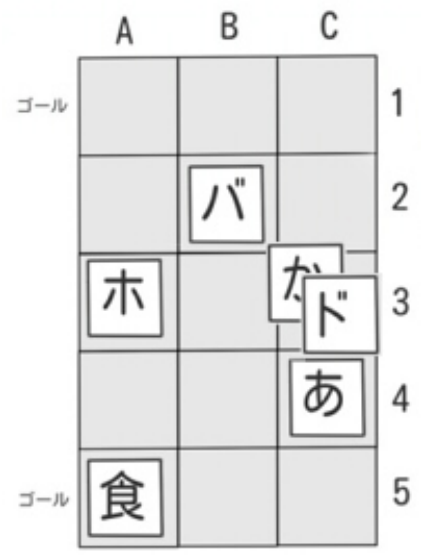


図 22:5.カ C3+, ド xC3#

4.1.3 1. ア C4, バ B2 の場合

1. ア C4, バ B2 の場合、先手が ア B4、食 B5, カ A4 は全て 2. ...ド C2 を指されて負けとなる。2. カ B4 は 2...ド C2 となり、2 カ B4 は 2.ド C2 で前述の 1. ア B4, バ A2, 2. カ A4, ド B2 先手後手を入れ替えた形になる。以下 、

3. 食 B5, ホ A2,

4. 食 A5, ホ A3

と進む(図 19)。5 手目でカレ―パンマンを動かす手は 、

5. カ A3, バ A3 (図 20)

でバイキンマンのゴールが確定するか、

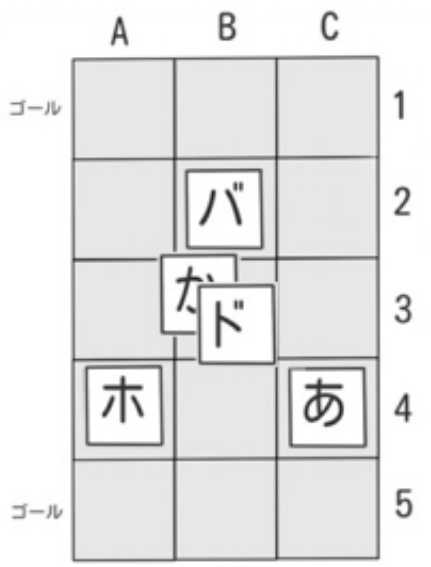


図 23:5.食 A4,ホ xA4,6.カ B3+,ド xB3#

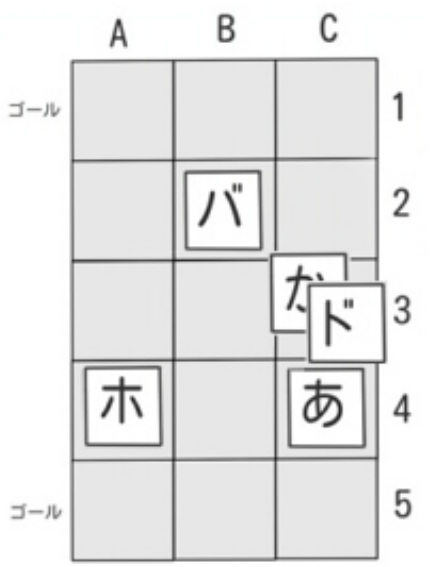


図 24:5.食 A4,ホ xA4,6.カ C3+,ド C3#

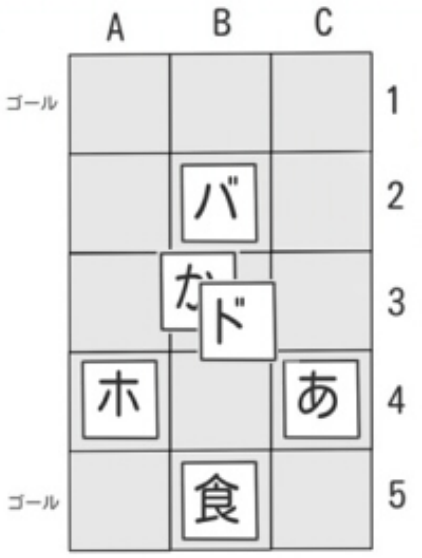


図 25:6.食 A5,ホ xB4+,7.ア xB4,ド B3#

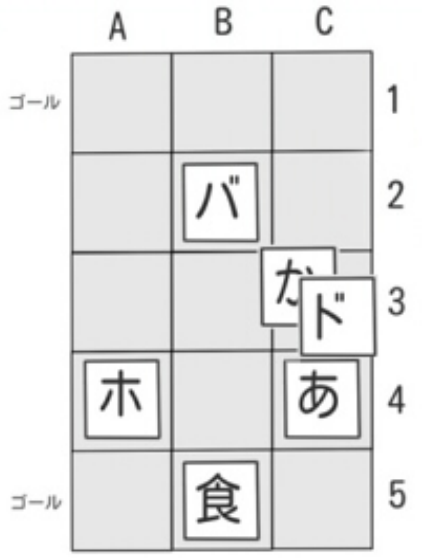


図 26:6.食 C5,ホ xB4+,7.ア xB4,ド B3#

5. カ B3+, ド xB3#（図 21）

5. カ C3+, ド xC3#（図 22）

で詰みとなる。 5 手目で食パンマンを動かす手は 5. 食 A4 は、

5. 食 A4, ホ xA4, 6. カ B3+, ド xB3#（図 23）

5. 食 A4, ホ xA4, 6. カ C3+, ド xC3#（図 24）

で詰み、5. 食 B5 と逃げても、

5. 食 B5, ホ A4, 6. 食 A5. ホ xB4+, 7. ア xB4. ド B3#（図 25）

5. 食 B5, ホ A4, 6. 食 C5, ホ xB4+, 7. ア xB4, ド B3#（図 26）

で詰みとなる。よって、1. ア C4, バ B2, は、7 手で後手勝ちである。

4.1.4 その他の場合の考察

アンパンマン、バイキンマンどちらかが初期置から１列手前に動かないときは有効な着手可能手が多く存在するので数えることは不可能である。しかしある程度の傾向は存在する。互いにリーダーを前進させると、着手可能手が減ることから、ツークツワンクが発生し易くなり、そこから勝負が付きやすい。ツークツワンクとはゲーム理論用語で、手番であることゲーム結果が悪化するのでパスしたい局面の事である。

リーダーの初期位置から前のマスを空けることで負け局面を回避できる場面が多く存在する事から均衡状態から我慢比べとなり千日手の引き分けが両者最善手であると推測する。

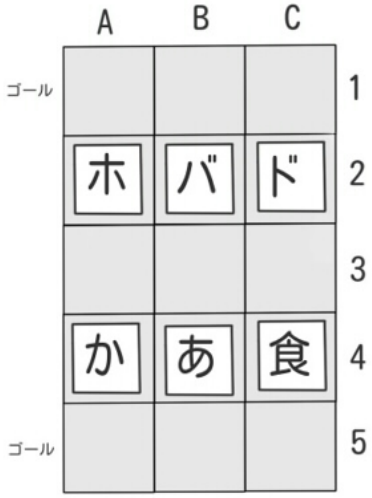


図 27:均衡状態

4.2 計算機実験の結果

本研究で作成した AI 同士の対戦を先読み手数 5 手で 100 回行ったところ、先手の 33 勝 53 負 14 引き分けであった。AI 同士の対戦により将譜、詰み局面から後手有利であること推測される。

5　結論および今後の課題

本研究ではアンパンマン将棋の完全解析の前準備としてアンパンマン将棋 AI 開発と計算機実験および限定された局面での部分解析を行った。AI 同士を対戦させる計算機実験の結果からは、後手有利と推測される。

部分解析の結果先手はリーダーA4、C4 は後手必勝ということがわかった。また、先手がリーダーB4 を指すと、後手は有効手が少なく、初手リーダー以外をを指すと、先手のリーダーは前進せずリーダー手前を空けることで手番を調整できる。そうなると後手側は有効手がなく千日手になると推測する。結果、両者最善を尽くすと引き分けになると推測する。しかし引き分けを証明するには全ての局面を調べる必要がありコンピュータ解析は必ず必要となる。よって今後の課題として、動物将棋に倣って全ての局面を数え上げた、後退解析（retrograde analysis）により，すべての局面勝ち、負け、引き分けいずれか３値に決定する必要がある。

計算機実験からは後手有利、部分解析から初手リーダーB4 で先手有利、というのは互いに矛盾しているが、バイキンマンのほうが図 27 の様な均衡状態（動くと負け）になったとき後手が多いからである。コンピュータ解析を行うことで、アンパンマン初手が必勝な方法がある場合は考えられるがバイキンマン後手側のほうがその可能性は高いと推論したためである。

謝辞

本研究をするにあたり、終始適切な助言を賜り、また丁寧に指導して下さった石水隆講師に感謝します。

同じ内容の研究を行なっている西川千明さんや、研究室のメンバーには常に刺激的な議論を頂き、精神的にも支えられました。ありがとうございます。

皆様に心から感謝します。本当にありがとうございました。

参考文献

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

[2] 岸本章宏, 柴原一友, 鈴木 豪, 小谷善行, ゲーム計算メカニズム-将棋･囲碁･オセロ･チェスのプログラムはどう動く-:pp2-20, コロナ社, (2010).

[3] 池泰弘,-Java 将棋のアルゴリズム:工学社, (2007).

[4] 池 泰弘, コンピュータ将棋のアルゴリズム―最強アルゴリズムの探求とプログラミング,工学社, (2005)

[5] 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/

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

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

[8] Joel Feinstein, Amenor Wins World 6x6 Championships!, Forty billion noted under the tree (July 1993), pp.6-8, British Othello Federation's newsletter., (1993), http://www.britishothello.org.uk/fbnall.pdf

[9] 清慎一, 川嶋俊, 探索プログラムによる四路盤囲碁の解, 情報処理学会研究報告, Vol. 2000-GI-004, pp.69-76, (2000), http://id.nii.ac.jp/1001/00058633/

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

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

[12] 「ごろごろどうぶつしょうぎ」発売開始！, お知らせ, 日本将棋連盟, 2012 年 11 月 26 日, (2012), http://www.shogi.or.jp/topics/2012/11/post-652.html

[13] 北尾まどか, 藤田麻衣子, どうぶつしょうぎねっと, (2010), http://dobutsushogi.net/

[14] IBM100 – Deep Blue, IBM, (1997), http://www-03.ibm.com/ibm/history/ibm100/us/en/icons/deepblue/

[15] Michael Khodarkovsky and Leonid Shamkvoich, 人間対機械－チェス世界チャンピオンとスーパーコンピューターの闘いの記録, 毎日コミュニケーションズ, (1998)

[16] 伊藤英紀, A 級リーグ差し手 1 号, (2013), http://aleag.cocolog-nifty.com/

[17] 米長邦雄, われ敗れたり コンピュータ棋戦のすべてを語る, 中央公論社, (2012).

[18] 日本囲碁規約逐条解説　第十二条

1. http://www.nihonkiin.or.jp/joho/kiyaku/kiyaku.htm

[19] 岸本章宏, 柴原一友, 鈴木 豪, 小谷善行, ゲーム計算メカニズム-将棋･囲碁･オセロ･チェスのプログラムはどう動く-:pp.21-22, コロナ社, (2010)

[20] 田中哲郎,ボードゲーム「シンペイ」の完全解析, 情報処理学会研究報告 Vol. 2006(23), pp.65-72(2006) http://ci.nii.ac.jp/naid/110004683809

[21] 高橋 大介, 佐藤 佳州, 2U-4 モンテカルロ法によるコンピュータ将棋の実現(ゲーム・知識ベース,学生セッション,人工知能と認知科学) 全国大会講演論文集 第 70 回平成 20 年, No.2, pp.123-124, 2008-03-13 http://ci.nii.ac.jp/naid/110006865370

[22] オセロプログラムと人間はどっちが強いのか？ロジステロとの戦い

http://uguisu.skr.jp/othello/7-2.html

[23] Michael Buro , LOGISTELLO, 2002, https://skatgame.net/mburo/log.html

付録

以下に本研究で作成したプログラムを添付する。

クラス.Anpanman

```
package anpanman;

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

public class Anpanman {
	final static int ANPANMAN = 1; _u32 ?8 // アンパンマン
	final static int SHOKUPANMAN = 2;@ // 食パンマン
	final static int CURRYPANMAN = 3;_u32 ?// カレーパンマン
	final static int BAIKINMAN = -1; P // バイキンマン
	final static int HORRORMAN = -2; X // ホラーマン
	final static int DOKINCHAN = -3; _u32 ?// ドキンちゃん
	final static int EMPTY = 0;    _u32 ?h // 空白
	final static int BORDER = Integer.MAX_VALUE; // 盤外外
	static FileWriter pass,_u114 ?Pass;  _  // 棋譜出力用
	static PrintWriter fp, rfp;



	public static void main (String[]_u97 ?rgs) {
		Board board =_cf2 new Board();
		boolean isCom[] = {false, false};
		//Scanner keyBoardScanner = new Scanner(System.in);		人間対戦時必要
		//String input;_u47 ?/ 入力用;
	人間対戦時必要
		String score;_cf11 // 棋譜;
		FileWriter pass≒ null; // 棋譜出力用ファイル
		PrintWriter fp = null;  // 棋譜出力用ファイルのポインタ
		try {
			pass = new FileWriter ("anpanmanScore.txt");
			fp = new PrintWriter (pass);
		} catch (IOException e) {
			System.err.println (e);
		}
```

```
        /*コンピュータに委託するか、人間が持つか決める：
人間対戦時必要
        System.out.print ("アンパンマンチームはCOMが持ちますか？ (Y/N) ");
        input = keyBoardScanner.next();
        if (input.equals ("Y") || input.equals ("y")) {
                isCom[0] = true;
        }
        System.out.print ("バイキンマンチームはCOMが持ちますか？ (Y/N) ");
        input = keyBoardScanner.next();
        if (input.equals ("Y") || input.equals ("y")) {
                isCom[1] = true;
        }*/
        //------------------------------------------------
        //全てコンピュータが行う
        isCom[0]=true;
        isCom[1]=true;
        //------------------------------------------------
        while (true) {
                board.showBoard();
                board.createMovableList(0); // アンパンマンチームが移動可能な手を求めるü

                System.out.println (board.value(1));

                if (board.isCheckedE(0)) System.out.println ("王手！");
                if (board.checkWin (1)) break;
                if (isCom[0]) {
                        score = board.com (0);
                } else {
                        score = board.player (0);
                }
                fp.print (score); // 棋譜出力



                board.showBoard();
                board.createMovableList%(1); // バイキンマンチームが移動可能な手を求める
                System.out.println (board.value(0));
                if (board.isChecked  u49 ?)) System.out.println ("王手！");
                if (board.checkWin (0)) break;
                if (isCom[1]) {
```

```
                    score = board.com (1);
                } else {
                    score = board.player (1);
                }
                fp.println (score); // 棋譜出力
            }
            fp.close();
        }
}
```

クラス.Board

```
package anpanman;

import java.util.Scanner;
import java.util.ArrayList;
import java.util.Random;

public class Board {
        final static int 餡 = 1;   // アンパンマン
        final static int 食 = 2;  // 食パンマン
        final static int 華 = 3;  // カレーパンマン
        final static int 菌 = -1; // バイキンマン
        final static int 骨 = -2; // ホラーマン
        final static int ど = -3; // ドキンちゃん
        final static int 空 = 0;  // 空白
        final static int 外 = Integer.MAX_VALUE; // 盤外


        public int [][][] sennichi;
        public int time;

        final static boolean isChessStyleScore = false; // 棋譜表記をチェス式か将棋式か？

        public int[][] board // 5*7の将棋盤
                        = {{外,外,外,外,外},
                           {外,骨,菌,ど,外},
                           {外,空,空,空,外},
                           {外,空,空,空,外},
                           {外,空,空,空,外},
                           {外,華,餡,食,外},
                           {外,外,外,外,外}};
        public int[][] boardclone;

        public  ArrayList<Integer> boardcopy = new ArrayList<Integer>();//boardcopy

        int 移動先のX座標; // 移動先のX座標//nextFile
        int 移動先のY座標; // 移動先のY座標//nextRank
        Piece anpanman, shokupanman, currypanman, baikinman, horrorman, dokinchan; // 駒
```

```
Boolean resign_u61 ?Efalse; // 投了したか　　すべてtrue になると負け
Boolean checkmate = false; // 詰んだ(チェックメイト)か
Boolean stalemate = false; // 詰んだ(スティールメイト)かH
ArrayList<NextMove> movableList;  // 候補手のリスト
int maxDepth = 5; // 先読みする手数の上限;

/**
 * コンストラクタ
 * 盤上に駒を初期設定で生成
 */
public Board () {
        anpanman    =_cf2 new Piece (餡);
        shokupanman =_cf2 new Piece (食);
        currypanman =_cf2 new Piece (華);
        baikinman   =_cf2 new Piece (菌);
        horrorman   =_cf2 new Piece (骨);
        dokinchan   =_cf2 new Piece (ど);
}

/**
 * コンストラクタ
 * 盤上に駒を指定した位置に配置
 * @param int[][] board 現在の盤
 */
public Board (int[][] board) {
        for (int r = 1; r <= 5; ++r)
                for (int f = 1; f <= 3; ++f)
                        this.board[r][f] = board[r][f];
        for (int r = 1 ; r <= 5; ++r) {
                for (int f = 1; f <= 3; ++f) {
                        switch (board [r][f]) {
                        case 餡 :
                                anpanman =_cf2 new Piece (餡, f, r);
                                break;
                        case 食 :
                                shokupanman = new Piece (食, f, r);
                                break;
                        case 華 :
                                currypanman = new Piece (華, f, r);
```

```
                                        break;
                                case 菌 :
                                        baikinman = new Piece (菌, f, r);
                                        break;
                                case 骨 :
                                        horrorman = new Piece (骨, f, r);
                                        break;
                                case ど :
                                        dokinchan = new Piece (ど, f, r);
                                        break;
                                }
                        }
                }
        }

        /**
         * 千日手をチェックする
         */
        public boolean checkSennichi() {
                int count=0;
                int check=0;
                for (int x = 0; x < 5; x++) {
                        for (int y = 0; y < 7; y++) {
                                sennichi[y][x][time] = board[y][x];
                        }
                }
                for (int i = 0; i < time; i++) {
                        count=0;
                        for (int x = 0; x < 5; x++) {
                        for (int y = 0; y < 7; y++) {
                                if (sennichi[y][x][i] == board[y][x]) {
                                count++;
                                }
                        }
                }
                if(count==35){
                        check++;
                }
        }
```

```
System.out.println(check);
if(check>=3){
          System.out.println("千日手のため引き分け");
          return true;
}
return false;
}

/**
 * 盤を表示
 */
public void showBoard () {
          System.out.println ("  ");
          for (int r = 0; r < board.length; ++r) {

                    for (int f = 0; f < board[r].length; ++f) {
                              System.out.print (showPiece (board[r][f]));
                    }
                    System.out.println();
          }
}

/**
 * 駒を表示
 * @param 駒の種類
 * @return 駒の文字列表現
 */
public String showPieceG(int type) {
          switch (type) {
          case 餡 :
                    return "餡";
          case 食 :
                    return "食";
          case 華 :
                    return "華";
          case 菌 :
                    return "菌";
          case 骨 :
                    return "骨";
```

```
        case ど :
                return "ど";
        case 空 :
                return "○";
        case 外 :
                return "";
        default :
                return "？";
        }
}

/**
 * 各プレイヤーの手番
 * @paran int playerNum プレイヤー番号
 * @return 指した手の棋譜
 */
public String player (int playerNum) {
        Scanner keyBoardScanner = new Scanner(System.in);
        String inputPiece;    // 駒の種類入力用
        Piece piece;      // 動かす駒
        int type;         // 動かす駒の種類
        int nextFile, nextRank; // 移動先
        ArrayList<NextMove> onesMovableList; // ある駒が移動可能な手のリスト

        while (true) { // 適切な駒が選択されるまでループ
                if (playerNum == 0)@{
                        System.out.println ("アンパンマンチームの番です");
                        System.out.print ("進める駒(A,S,C)を選んでください(R=投了):");
                } else {
                        System.out.println ("バイキンマンチームの番です");
                        System.out.print ("進める駒(B,H,D)を選んでください(R=投了):");
                }
                inputPiece =_u107 ?eyBoardScanner.next();
                if (inputPiece.equals ("A") || inputPiece.equals ("a")) {
                        piece = anpanman;
                        type = 餡;
                } else if (inputPiece.equals ("S") || inputPiece.equals ("s")) {
                        piece = shokupanman;
                        type = 食;
```

```
} else if (inputPiece.equals ("C") || inputPiece.equals ("c")) {
        piece = currypanman;
        type = 華;
} else if (inputPiece.equals ("B") || inputPiece.equals ("b")) {
        piece = baikinman;
        type = 菌;
} else if (inputPiece.equals ("H") || inputPiece.equals ("h")) {
        piece = horrorman;
        type = 骨;
} else if (inputPiece.equals ("D") || inputPiece.equals ("d")) {
        piece = dokinchan;
        type = ど;
} else if (inputPiece.equals ("R") || inputPiece.equals ("r")) {
        System.out.println ("投了します");
        resign = true;
        if (isChessStyleScore) {
                if (playerNum == 0)0{
                        return "1-0";
                } else {
                        return "0-1";
                }
        } else {
                return "投了";
        }
} else {
        if (playerNum == 0)x{
                System.out.println ("A,S,C から選んでください");
        } else {
                System.out.println ("B,H,D から選んでください");
        }
        continue; // 選び直し
}

if (playerNum == 0 && !(type == 餡 || type == 食 || type == 華)) {
        System.out.println ("A,S,C から選んでください");
        continue; // 選び直し
} else if (playerNum == 1 && !(type == 菌 || type == 骨 || type == ど)) {
        System.out.println ("B,H,D から選んでください");
        continue; // 選び直し
```

```
                    }

                    if (piece == null) {
                              System.out.println (name (type) + "はすでに取られています");
                              continue; // 選び直し
                    }

                    onesMovableList = new ArrayList<NextMove>(); // 指定した駒が移動可能な手のリスト
                    for (int i = 0;  i < movableList.size(); ++i) {
                              if (movableList.get(i).type() == type) {       // 指定した駒を動かす手か？

                                        onesMovableList.add (movableList.get(i));  // 指定した駒が移動可能
な手の数を加える
                              }
                    }
                    if (onesMovableList.size() == 0) { // 指定した駒が移動可能な位置が無い場合
                              System.out.println (name (type) + "が進める位置はありません");
                              continue; // 選び直し
                    }
                    System.out.println (name (type) + "が進む場所を選んでください");
                    System.out.print (name (type) + "は現在(" + piece.file() + "," + piece.rank() + ")にい
て");
                    for (int i = 0; i < onesMovableList.size(); ++i) {
                              System.out.print ("(" +  onesMovableList.get(i).nextFile() + "," +
onesMovableList.get(i).nextRank() + ")");
                    }
                    System.out.println ("へ移動できます");

                    System.out.print ("x座標 (1～3):");
                    nextFile = keyBoardScanner.nextInt();
                    System.out.print ("y座標 (1～5):");
                    nextRank = keyBoardScanner.nextInt();

                    boolean movable = false; // 指定した位置に移動可能か
                    for (int i = 0; i < onesMovableList.size(); ++i) {
                              if ((nextFile == onesMovableList.get(i).nextFile())
                              && (nextRank == onesMovableList.get(i).nextRank())) {
                                        movable = true;
                                        break;
```

```
                }
            }
            if (!movable) {
                System.out.println (name (type) + "は(" + nextFile + "," + nextRank + ")へは
移動できません");
                continue; // 選び直し
            }
            break; // while ループから脱出
        }
        return movePiece (piece, type, nextFile, nextRank); // 駒を移動させる
    }

    /**
     * COMの手番
     * @paran int playerNum プレイヤー番号
     * @return 指した手の棋譜
     */
    public String com (int playerNum) {
        int type, nextFile, nextRank;      // 移動する駒の種類, 移動先の座標
        Piece piece = null;              // 移動する駒
        int bestValue;                   // 最も良い盤面の評価値
        NextMove bestMove = null;        // 最も良い手
        int nextPlayerNum;               // 次の手番プレイヤー

        if (playerNum == 0)ﾀ{ // アンマンパンチームの場合_u-30123 ?価値が高いほど良い手と見做す
            nextPlayerNum = 1;          // 次の手番プレイヤー
            bestValue = Integer.MIN_VALUE;
            for (int i=0; i<movableList.size(); ++i) {
                NextMove nextMove = movableList.get(i);    // i番目の候補手
                Board nextBoard = nextBoard (nextMove, nextPlayerNum);    // 次の盤面を生成
                int value = nextBoard.value (nextPlayerNum, maxDepth);// 盤面の評価値を計算
                if (value > bestValue) {                  // 高評価の手を発見した
                    bestMove =_u110 ?extMove;                  // 高評価の手を記憶
                    bestValue = value;                    // 評価値を記憶
                }
                if (bestValue == Integer.MAX_VALUE) {    // 評価無限大の手＝必勝の手を発見
                    break;              _u32 ?c              // ループ脱出
                }
            }
```

```
            if (bestValue == Integer.MIN_VALUE) {  // 評価値無限小の手しか無い＝負け確定
                    System.out.println ("投了します");
                    resign = true;
                    if (isChessStyleScore) {
                            return "1-0";
                    } else {
                            return "投了";
                    }
            }
    } else { // バイキンマンチームの場合_u-30123 ?価値が低いほど良い手と見做す
            nextPlayerNum = 0;          // 次の手番プレイヤー
            bestValue = Integer.MAX_VALUE;
            for (int i=0; i<movableList.size(); ++i) {
                    NextMove nextMove = movableList.get(i);    // i番目の候補手
                    Board nextBoard = nextBoard (nextMove, nextPlayerNum);    // 次の盤面を生成
                    int value = nextBoard.value (nextPlayerNum, maxDepth);// 盤面の評価値を計算
                    if (value < bestValue) {                  // 高評価の手を発見した
                            bestMove =_u110 ?extMove;                  // 高評価の手を記憶
                            bestValue = value;                    // 評価値を記憶
                    }
                    if (bestValue == Integer.MIN_VALUE) {    // 評価無限小の手＝必勝の手を発見
                            break;              _u32 ?ｵ              // ループ脱出
                    }
            }
            if (bestValue == Integer.MAX_VALUE) {  // 評価値無限大の手しか無い＝負け確定
                    System.out.println ("投了します");
                    resign = true;
                    if (isChessStyleScore) {
                            return "0-1";
                    } else {
                            return "投了";
                    }
            }
    }
    type = bestMove.type();          // 移動する駒の種類
    nextFile = bestMove.nextFile(); // 移動先のX座標
    nextRank = bestMove.nextRank(); // 移動先のY座標

    switch (type) {
```

```
            case 餡 : piece = anpanmanl    break;
            case 食 : piece = shokupanman; break;
            case 華 : piece = currypanman; break;
            case 菌 : piece = baikinman;   break;
            case 骨 : piece = horrorman;   break;
            case ど : piece = dokinchan;   break;
            }
            return movePiece (piece, type, nextFile, nextRank); // 駒を移動させる
}

/**
 * 指定した位置に駒を移動させ、その棋譜表記を返す
 * @param Piece piece 移動させる駒
 * @param int type 移動させる駒の種類
 * @param int nextFile 移動先のX座標
 * @param int nextRank 移動先のY座標
 */
public String movePiece(Piece piece, int type, int nextFile, int nextRank) {ø
            String score;_cf11 // 棋譜
            if (isChessStyleScore) { // チェス風の棋譜を作成
                        switch (type) {
                        case 餡 : score = "A"; break;
                        case 食 : score = "S"; break;
                        case 華 : score = "C";(break;
                        case 菌 : score = "B"; break;
                        case 骨 : score = "H"; break;
                        case ど : score = "D"; break;
                        default : score = "?"; break;
                        }
                        if (board[nextRank][nextFile] != 空) score += "x";
                        switch (nextFile) {
                        case 1 :  score += "a"; break;
                        case 2 :  score += "b"; break;
                        case 3 :  score += "c"; break;
                        default : score += "?"; break;
                        }
                        switch (nextRank) {
                        case 1 :  score += "1 "; break;
                        case 2 :  score += "2 "; break;
```

```
            case 3 :  score += "3 "; break;
            case 4 :  score += "4 "; break;
            case 5 :  score += "5 "; break;
            default : score += "? "; break;
            }
} else { // 将棋風の棋譜を作成
            switch (nextFile) {
            case 1 :  score = "１"; break;
            case 2 :  score = "２"; break;
            case 3 :  score = "３"; break;
            default : score = "？"; break;
            }
            switch (nextRank) {
            case 1 :  score += "一"; break;
            case 2 :  score += "二"; break;
            case 3 :  score += "三"; break;
            case 4 :  score += "四"; break;
            case 5 :  score += "五"; break;
            default : score += "？"; break;
            }
            switch (type) {
            case 餡 : score += "餡  "; break;
            case 食 : score += "食  "; break;
            case 華 : score += "華  "; break;
            case 菌 : score += "菌  "; break;
            case 骨 : score += "骨  "; break;
            case ど : score += "ど  "; break;
            default : score += "？  "; break;
            }
}
if (board[nextRank][nextFile] != 空) { // 移動先に駒がある場合
            removePiece (nextFile, nextRank); // 移動先にある駒を取り除く
}

board[piece.rank()][piece.file()] = 空; // 移動前のマスを空白に
piece.move (nextFile, nextRank);           // 駒を移動
board[piece.rank()][piece.file()] = type;  // 移動後のマスを指定した駒に

return score;
```

```
	}

	/**
	 * 指定した位置にある駒を盤から取り除く
	 * @param int file X座標
	 * @param int rank Y座標
	 */
	public void removePiece (int nextFile, int nextRank) {
		switch (board [nextRank][nextFile]) {
		case 餡 :                  // 移動先にアンパンマン
			anpanman = null;              // アンパンマンを取り除く
			break;
		case 食 :                 // 移動先に食パンマン
			shokupanman = null;           // 食パンマンを取り除く
			break;
		case 華 :                 // 移動先にカレーパンマン
			currypanman = null;            // カレーパンマンを取り除く
			break;
		case 菌 :                 // 移動先にバイキンマン
			baikinman = null;              // バイキンマンを取り除く
			break;
		case 骨 :                 // 移動先にホラーマンˆ
			horrorman = null;               // ホラーマンを取り除く
			break;
		case ど :                 // 移動先にドキンちゃん
			dokinchan = null;              // ドキンちゃんを取り除く
			break;
		}
	}
/*
	/**
	 * 勝負がついたか
	 * @param int playerNum プレイヤー番号
	 * @return 勝負がついたか
	 */
	public boolean checkWin (int playerNum) {
		if (playerNum == 0){                    // アンパンマンチームの手番
			if (baikinman == null) {            // バイキンマンを取った
				System.out.println ("アンパンマンチームの勝利！");
```

```
                return true;
        } else if (anpanman.rank()-== 1) {  // アンパンマンがゴールした
                System.out.println ("アンパンマンチームの勝利！");
                return true;
        } else if (isMate (1)) {_cf11 // バイキンマンが詰んだ
                if (isChecked (1)) { // バイキンマンに王手がかかっている
                        System.out.println ("チェックメイト！");
                        System.out.println ("アンパンマンチームの勝利！");
                } else {
                        System.out.println ("スティールメイト！");
                        System.out.println ("引き分けです");
                }
                return true;
        } else if (resign) {            // 投了した
                System.out.println ("バイキンマンチームの勝利！");
                return true;
        } else {
                return false;
        }
} else {                            // バイキンマンチームの手番
        if (anpanman == null) {           _cf11 // アンパンマンを取った
                System.out.println ("バイキンマンチームの勝利！");
                return true;
        } else if (baikinman.rank() == 5) { // バイキンマンがゴールした
                System.out.println ("バイキンマンチームの勝利！");
                return true;
        } else if (isMate (0)) {_cf11 // アンパンマンチームが詰んだ
                if (isChecked (0)) { // アンパンマンに王手がかかっている
                        System.out.println ("チェックメイト！");
                        System.out.println ("バイキンマンチームの勝利！");
                } else {
                        System.out.println ("スティールメイト！");
                        System.out.println ("引き分けです");
                }
                return true;
        } else if (resign) {    _       // 投了した
                System.out.println ("アンパンマンチームの勝利！");
                return true;
        } else {
```

```
                        return false;
                }
        }
}

/**
 * 現在王手がかかっているか
 * @param int playerNum プレイヤー番号
 * @return 王手がかかっているか
 */
public boolean isChecked (int playerNum) {
        int file, rank;
        if (playerNum == 0) {
                file = anpanman.file(); // アンパンマンの座標
                rank = anpanman.rank();
                if (board [rank][file-1] == 骨) return true;          // ホラーマンが王手
                else if (board [rank-1][file] == 骨) return true;     // ホラーマンが王手
                else if (board [rank][file+1] == 骨) return true;     // ホラーマンが王手
                else if (board [rank-1][file-1] == ど) return true;   // ドキンちゃんが王手
                else if (board [rank-1][file] == ど) return true;     // ドキンちゃんが王手
                else if (board [rank-1][file+1] == ど) return true;   // ドキンちゃんが王手
                else return false;
        } else {
                file = baikinman.file(); // バイキンマンの座標
                rank = baikinman.rank();
                if (board [rank][file-1] == 食) return true;        // 食パンマンが王手
                else if (board [rank+1][file] == 食) return true;   // 食パンマンが王手
                else if (board [rank][file+1] == 食) return true;   // 食パンマンが王手
                else if (board [rank+1][file-1] == 華) return true; // カレーパンマンが王手
                else if (board [rank+1][file] == 華) return true;   // カレーパンマンが王手
                else if (board [rank+1][file+1] == 華) return true; // カレーパンマンが王手
                else return false;
        }
}

/**
 * 詰みの判定
 * 移動可能な手が無ければ詰み(チェックメイトまたはスティールメイト)
 * @param int playerNum プレイヤー番号
```

```
     * @return 詰んだか
     */
    public boolean isMate (int playerNum) {
            return (movableList.size() == 0); // 可能な手が無ければ詰み
    }

    /**
     * 候補手の作成
     * また，移動可能な手のリストを movableList に保持する
     * @param int playerNum プレイヤー番号
     */
    public void createMovableList (int playerNum) { //アンパンマンチームの場合
            if (playerNum == 0) { // アンパンマンチームの場合
                    movableList = anpanman.movableList (board);      // アンパンマンが移動可能な手
                    if (shokupanman != null) {                       // 盤上にまだ食パンマンがある場合
                            movableList.addAll (shokupanman.movableList (board)); // 食パンマンが移動可能
な手
                    }
                    if (currypanman != null) {                       // 盤上にまだカレーパンマンがある場合
                            movableList.addAll (currypanman.movableList (board)); // カレーパンマンが移動
可能な手
                    }
                    if (isChecked(0)) { // アンパンマンに王手が掛かっている場合
                            //int nextPlayerNum = (playerNum == 0) ? 1 : 0;   // 次のプレイヤー番号
                            for (int i = 0; i < movableList.8ize();) {
                                    Board nextBoard = nextBoard (movableList.get(i), playerNum);
                                    if (nextBoard.isChecked (0)) { // 移動後も王手が掛かっている手は無
効
                                            movableList.remove(i);     // 移動後も王手が掛かっている
手を取り除く
                                    } else {
                                            ++i;
                                    }
                            }
                    }
            } else { //バイキンマンチームの場合
                    movableList = baikinman.movableList (board);        // バイキンマンが移動可能な手
                    if (horrorman != null) {                           // 盤上にまだホラーマンがある場合
                            movableList.addAll (horrorman.movableList (board));   // ホラーマンが移動可能
```

```
な手
                        }
                        if (dokinchan != null) {                        // 盤上にまだドキンちゃんがある場合
                                movableList.addAll(dokinchan.movableList(board));   // ドキンちゃんが移動可
能な手
                        }
                        if (isChecked(1)) { // バイキンマンに王手が掛かっている場合
                                for (int i = 0; i < movableList.size();) {
                                        Board nextBoard = nextBoard(movableList.get(i), playerNum);
                                        if (nextBoard.isChecked(1)) { // 移動後も王手が掛かっている手は無
効
                                                movableList.remove(i);     // 移動後も王手が掛かっている
手を取り除く
                                        } else {
                                                ++i;
                                        }
                                }
                        }
                }
        }

        /**
         * 現在の盤の評価値を表示
         * 先読みは行わず現在の盤面のみで評価する
         * @param int playerNum プレイヤー番号
         * @return 評価値
         */
        public int value(int playerNum) {
                checkmate = false;
                stalemate = false;

                /* 現時点ですでに詰んでいるかどうかのチェック */
                if (playerNum == 0) { // アンパンマンチームの手番
                        if (anpanman == null) {            // アンパンマンが取られた
                                checkmate = true;
                                return Integer.MIN_VALUE; // 評価値無限小
                        } else if (baikinman.rank() == 5) { // バイキンマンがゴールした
                                checkmate = true;
                                return Integer.MIN_VALUE; // 評価値無限小
```

```
            } else if (isMate (0)) { // アンパンマンチームが詰んだ
                    if (isChecked (0)) { // アンパンマンに王手がかかっている
                            checkmate = true;
                            return Integer.MIN_VALUE; // 評価値無限小
                    }  else { // ステールメイト
                            stalemate = true;
                            return Integer.MIN_VALUE; // 評価値0ではなく評価値無限小
                    }
            } else if (resign) {    _          // バイキンマンチームが投了した
                    return Integer.MAX_VALUE; // 評価値無限大
            }
    } else { // バイキンマンチームの手番
            if (baikinman == null) {             // バイキンマンが取られた
                    checkmate = true;
                    return Integer.MAX_VALUE; // 評価値無限大
            } else if (anpanman.rank()== 1) {  // アンパンマンがゴールしたa
                    checkmate = true;
                    return Integer.MAX_VALUE; // 評価値無限大
            } else if (isMate (1)) { // バイキンマンが詰んだ
                    if (isChecked (1)) { // バイキンマンに王手がかかっている
                            checkmate = true;
                            return Integer.MAX_VALUE; // 評価値無限大
                    } else { // ステールメイト
                            stalemate = true;
                            return Integer.MIN_VALUE; // 評価値0ではなく評価値無限小
                    }
            } else if (resign) {             // アンパンマンチームが投了した
                    return Integer.MIN_VALUE; // 評価値無限小
            }
    }

    /* 現時点ではまだ詰んでいない場合 */
    int value = 0;
    switch (anpanman.rank()) { // アンパンマンはゴールに近い方が高評価
    case 1:   // アンパンマンがゴールした場合
            return Integer.MAX_VALUE; // すでにチェックしているのでここに処理が移ることはありえな
い

    case 2:
            value += 13;
```

```
                break;
        case 3:
                value += 9;
                break;
        case 4:
                value += 7;
                break;
        case 5:
                value += 6;
                break;
        }
        if (shokupanman != null) {        // 盤上に食パンマンがある場合
                if (shokupanman.rank() == 1) // 食パンマンは端まで進むと価値が下がる
                        value += 2;
                else value += 4;
        }
        if (currypanman != null) {        // 盤上にカレーパンマンがある場合
                if (currypanman.rank() == 1) // カレーパンマンは端まで進むと価値が下がる
                        value += 0;
                else value += 3;
        }
        switch (baikinman.rank()) { // バイキンマンはゴールに近い方が高評価
        case 5:  // バイキンマンがゴールした場合
                return Integer.MIN_VALUE; // すでにチェックしているのでここに処理が移ることはありえな
い
        case 4:
                value -= 13;
                break;
        case 3:
                value -= 9;
                break;
        case 2:
                value -= 7;
                break;
        case 1:
                value -= 6;
                break;
        }
        if (horrorman != null) {        // 盤上にホラーマンがある場合
```

```
                if (horrorman.rank() == 5) // ホラーマンは端まで進むと価値が下がる
                        value -= 2;
                else value -= 4;
        }
        if (dokinchan != null) {        // 盤上にドキンちゃんがある場合
                if (dokinchan.rank() == 5) // ドキンちゃんは端まで進むと価値が下がる
                        value -= 0;
                else value -= 3;
        }
        if (playerNum == 0)ツ{
                value += movableList.size(); // 移動可能な手が多いほど高評価値
        } else {
                value -= movableList.size(); // 移動可能な手が多いほど低評価値
        }
        return value;
}
/**
 * 現在の盤の評価値を表示
 * @param int playerNum プレイヤー番号
 * @param int depth 先読みする手数
 * @return 評価値
 */
public int value (int playerNum, int depth) {
        createMovableList (playerNum);      // 移動可能な手のリストを生成

        int value = value (playerNum); // 先読み無しの現在の盤面の評価値を求める
        if (depth == 0) {
                return value;
        }
        if (checkmate || stalemate || resign) return value; // すでに詰んでいるときはそのまま値を返す

        int bestValue;                  // 最も良い評価値
        NextMove bestMove = null;           // 最も良い手
        int nextPlayerNum;              // 次の手番プレイヤー

        if (playerNum == 0){ // アンパンマンチームの場合評価値が高いほど良い手と見做す
                nextPlayerNum = 1;              // 次の手番プレイヤー
                bestValue = Integer.MIN_VALUE;      // 盤面の評価値の初期値
```

```
                for (int i=0; i<movableList.size(); ++i) {
                        NextMove nextMove = movableList.get(i);    // i番目の候補手
                        Board nextBoard = nextBoard (nextMove, nextPlayerNum);    // 次の盤面を生成
                        value = nextBoard.value (nextPlayerNum, depth-1);// 盤面の評価値を計算
                        if (value > bestValue) {                // 高評価の手を発見した
                                bestMove = nextMove;                // 高評価の手を記憶
                                bestValue = value;                    // 評価値を記憶
                        }
                        if (bestValue == Integer.MAX_VALUE) {    // 評価無限大の手＝必勝の手を発見
                                return Integer.MAX_VALUE;
                        }
                }
        } else { // バイキンマンチームの場合評価値が低いほど良い手と見做す
                nextPlayerNum = 0;              // 次のプレイヤー番号
                bestValue = Integer.MAX_VALUE; // 盤面の評価値の初期値

                for (int i=0; i<movableList.size(); ++i) {
                        NextMove nextMove = movableList.get(i);    // i番目の候補手
                        Board nextBoard = nextBoard (nextMove, nextPlayerNum);    // 次の盤面を生成
                        value = nextBoard.value (nextPlayerNum, depth-1);// 盤面の評価値を計算
                        if (value < bestValue) {               // 高評価の手を発見した
                                bestMove = nextMove;                // 高評価の手を記憶
                                bestValue = value;                    // 評価値を記憶
                        }
                        if (bestValue == Integer.MIN_VALUE) {    // 評価無限小の手＝必勝の手を発見
                                return Integer.MIN_VALUE;
                        }
                }
        }
        Random rnd = new Random();
        int ran = rnd.nextInt (3); // 0～2の乱数を生成
        value = bestValue + ran - 1; // 評価値に乱数を加える
        return value;
}

/**
 * 指定した駒を指定した位置に動かした後の盤を得る
 * @param NextMove nextMove 次の手
 * @param int playerNum プレイヤー番号
```

```
 * @return 移動した後の盤
 */
public Board nextBoard (NextMove nextMove, int playerNum) {
        Board nextBoardh= new Board (board);
        int movingType =  nextMove.type(); // 移動する駒の種類
        Piece movingPiece = null; // 移動する駒
        switch (movingType) {
        case 餡 :
                movingPiece = nextBoard.anpanman;
                break;
        case 食 :
                movingPiece = nextBoard.shokupanman;
                break;
        case 華 :
                movingPiece = nextBoard.currypanman;
                break;
        case 菌 :
                movingPiece = nextBoard.baikinman;
                break;
        case 骨 :
                movingPiece = nextBoard.horrorman;
                break;
        case ど :
                movingPiece = nextBoard.dokinchan;
                break;
        }
        int currentFile = movingPiece.file(); // 移動する駒の現在のX座標
        int currentRank = movingPiece.rank(); // 移動する駒の現在のY座標
        int nextFile = nextMove.nextFile(); // 移動先のX座標
        int nextRank = nextMove.nextRank(); // 移動先のY座標
        nextBoard.movePiece (movingPiece, movingType, nextFile, nextRank); // 駒を移動させる

        return nextBoard;
}

/**
 * 駒名を返す
 * @param int type 駒の種類
 * @return 駒名
```

```
	 */
	public String name (int type) {
		switch (type) {
		case 餡 :
			return "アンパンマン";
		case 食 :
			return "食パンマン";
		case 華 :
			return "カレーパンマン";
		case 菌 :
			return "バイキンマン";
		case 骨 :
			return "ホラーマン";
		case ど :
			return "ドキンちゃん";
		default :
			return "？";
		}
	}
}
```

クラス.NextMove

```
package anpanman;

/**
 * 駒の移動可能な位置を表すクラス
 */
public class NextMove {
	final static int 餡 = 1;  // アンパンマン
	final static int 食 = 2;  // 食パンマン
	final static int 華 = 3;  // カレーパンマン
	final static int 菌 = -1; // バイキンマン
	final static int 骨 = -2; // ホラーマン
	final static int ど = -3; // ドキンちゃん

	int type;      // 駒の種類
	int nextFile;  // 移動先のX座標
	int nextRank;  // 移動先のY座標
	int value;     // 移動した場合の盤面の評価値

	/**
	 * コンストラクタ
	 * @param int type 駒の種類
	 * @param int nextFile 移動先のX座標
	 * @param int nextRank 移動先のY座標
	 */
	public NextMove (int type, int nextFile, int nextRank) {
		this.type = type;
		this.nextFile = nextFile;
		this.nextRank = nextRank;
	}

	/**
	 * 駒の種類を返す
	 * @return 駒の種類
	 */
	public int type() {
		return type;
	}
```

```
	/**
	 * 移動先のX座標を返す
	 * @return X座標
	 */
	public int nextFile() {
		return nextFile;
	}

	/**
	 * 移動先のY座標を返す
	 * @return Y座標
	 */
	public int nextRank() {
		return nextRank;
	}


	/**
	 * 移動した場合の盤面の評価値を返す
	 * @return 評価値
	 */
	public int value() {
		return value;
	}

	/**
	 * 評価値をセットする
	 * @param value 評価値
	 */
	public void setValue (int value) {
		this.value = value;
	}
}
```

クラス.Piece

```
package anpanman;

import java.util.ArrayList;

public class Piece {
	final static int ANPANMAN = 1;	// アンパンマン
	final static int SHOKUPANMAN = 2;	// 食パンマン
	final static int CURRYPANMAN = 3;	// カレーパンマン
	final static int BAIKINMAN = -1;	// バイキンマン
	final static int HORRORMAN = -2;	// ホラーマン
	final static int DOKINCHAN = -3;	// ドキンちゃん
	final static int EMPTY = 0;		// 空白
	final static int BORDER = Integer.MAX_VALUE; // 盤外

	final static int[] アンパンマンの移動可能X方向	= {-1,-1, 0, 1, 1}; // アンパンマンの移動可能X方向
	final static int[] アンパンマンの移動可能Y方向	= { 0,-1,-1,-1, 0}; // アンパンマンの移動可能Y方向
	final static int[] 食パンマンの移動可能X方向 = {-1, 0, 1};		// 食パンマンの移動可能X方向
	final static int[] 食パンマンの移動可能Y方向 = { 0,-1, 0};		// 食パンマンの移動可能Y方向
	final static int[] カレーパンマンの移動可能X方向 = {-1, 0, 1};		// カレーパンマンの移動可能X方向
	final static int[] カレーパンマンの移動可能Y方向 = {-1,-1,-1};		// カレーパンマンの移動可能Y方向
	final static int[] バイキンマンの移動可能X方向	= {-1,-1, 0, 1, 1}; // バイキンマンの移動可能X方向
	final static int[] バイキンマンの移動可能Y方向	= { 0, 1, 1, 1, 0}; // バイキンマンの移動可能Y方向
	final static int[] ホラーマンの移動可能X方向	= {-1, 0, 1};		// ホラーマンの移動可能X方向
	final static int[] ホラーマンの移動可能Y方向	= { 0, 1, 0};		// ホラーマンの移動可能Y方向
	final static int[] ドキンちゃんの移動可能X方向	= {-1, 0, 1};		// ドキンちゃんの移動可能X方向
	final static int[] ドキンちゃんの移動可能Y方向	= { 1, 1, 1};		// ドキンちゃんの移動可能Y方向

	boolean isOnBoard; // 盤上に駒があるか
	int X軸;		// 駒のX座標
	int Y軸;		// 駒のY座標
	int type;		 // 駒の種類
	int movableRankVector[], movableFileVector[];	// 移動可能方向(駒の現在位置を(0,0)とした場合の相対位置)

	/**
	 * コンストラクタ
	 * 駒を生成し初期位置に置く
	 * @param int type 駒の種類
```

```
 */
public Piece (int type) {
          this.type = type;
          setMovableVector();
          setInitialPosition();
}

/**
 * コンストラクタ
 * 駒を生成し指定した位置に置く
 * @param int type 駒の種類
 * @param int
 */
public Piece (int type, int file, int rank) {
          this.type = type;
          setMovableVector();
          this.X軸 = file;
          this.Y軸 = rank;
          this.isOnBoard = true;
}

/**
 * 駒の移動可能方向をセット
 */
private void setMovableVector() {
          switch (type) { // 駒の種類により分岐
          case ANPANMAN : // アンパンマンの場合
                    movableFileVector = アンパンマンの移動可能X方向;
                    movableRankVector = アンパンマンの移動可能Y方向;
                    break;
          case SHOKUPANMAN : // 食パンマンの場合
                    movableFileVector = 食パンマンの移動可能X方向;
                    movableRankVector = 食パンマンの移動可能Y方向;
                    break;
          case CURRYPANMAN : // カレーパンマンの場合
                    movableFileVector = カレーパンマンの移動可能X方向;
                    movableRankVector = カレーパンマンの移動可能Y方向;
                    break;
          case BAIKINMAN : // バイキンマンの場合
```

```
                movableFileVector = バイキンマンの移動可能X方向;
                movableRankVector = バイキンマンの移動可能Y方向;
                break;
        case HORRORMAN : // ホラーマンの場合
                movableFileVector = ホラーマンの移動可能X方向;
                movableRankVector = ホラーマンの移動可能Y方向;
                break;
        case DOKINCHAN : // ドキンちゃんの場合
                movableFileVector = ドキンちゃんの移動可能X方向;
                movableRankVector = ドキンちゃんの移動可能Y方向;
                break;
        default : // それ以外の場合
                System.out.println ("駒の種類を認識できません");
        }
}

/**
 * 駒を盤上の初期位置にセット
 */
private void setInitialPosition(){
        switch (type) {
        case ANPANMAN :    // アンパンマンの場合
                Y軸 = 5;
                X軸 = 2;
                break;
        case SHOKUPANMAN : // 食パンマンの場合
                Y軸 = 5;
                X軸 = 3;
                break;
        case CURRYPANMAN : // カレーパンマンの場合
                Y軸 = 5;
                X軸 = 1;
                break;
        case BAIKINMAN :   // バイキンマンの場合
                Y軸 = 1;
                X軸 = 2;
                break;
        case HORRORMAN :   // ホラーマンの場合
                Y軸 = 1;
```

```
                X軸 = 1;
                break;
        case DOKINCHAN :   // ドキンちゃんの場合
                Y軸 = 1;
                X軸 = 3;
                break;
        }
        isOnBoard = true;
}

/**
 * 駒を盤上の指定した座標にセット
 * @param int file X座標
 * @param int rank Y座標
 */
private void setPosition (int x, int y){
        this.X軸 = x;
        this.Y軸 = y;
        isOnBoard = true;
}

/**
 * 駒を盤上に置く
 */
public void setOnBoard() {
        isOnBoard = true;
}

/**
 * 駒を盤上から削除
 */
public void removeFromBoard() {
        isOnBoard = false;
}

/**
 * 盤上に駒が存在するか
 * @return 駒が存在するか
 */
```

```
public boolean isOnBoard() {
	return isOnBoard;
}

/**
 * 指定した座標に駒が存在するか
 * @param int file x座標
 * @param int rank y座標
 * @return 駒が存在するか
 */
public boolean isThere (int file, int rank) {
	if (isOnBoard) {
		return ((this.X軸 == file) && (this.Y軸 == rank));
	} else return false;
}

/**
 * x座標を返す
 * @return x座標
 */
public int file(){
	return this.X軸;
}

/**
 * y座標を返す
 * @return y座標
 */
public int rank(){
	return this.Y軸;
}

/**
 * 駒の種類を返す
 * @return 駒の種類
 */
public int type(){
	return this.type;
}
```

```
/**
 * 駒名を返す
 * @return 駒名
 */
public String name() {
	switch (type) {
	case ANPANMAN :
		return "アンパンマン";
	case SHOKUPANMAN :
		return "食パンマン";
	case CURRYPANMAN :
		return "カレーパンマン";
	case BAIKINMAN :
		return "バイキンマン";
	case HORRORMAN :
		return "ホラーマン";
	case DOKINCHAN :
		return "ドキンちゃん";
	default :
		return "？";
	}
}

/**
 * 駒を指定した座標に移動する
 * @param nextFile 移動するX座標
 * @param nextRank 移動するY座標
 */
public void move (int nextFile,int nextRank) {
	X軸 = nextFile;
	Y軸 = nextRank;
}

/**
 * 駒が指定した座標に移動できるか
 * 他の駒は無視して自分が移動できるかのみを判断する
 * @param int nextFile 移動したいX座標
 * @param int nextRank 移動したいY座標
```

```
 * @return 移動できるか
 */
public boolean movable (int nextFile,int nextRank) {
	if (!isOnBoard) // すでに取られている場合
		return false;
	if (nextFile < 1 || nextFile > 3 || nextRank < 1 || nextRank > 5) // 盤外を指定した場合
		return false;
	for (int i = 0; i < movableFileVector.length; ++i) {
		if ((nextFile == X軸 + movableFileVector[i]) && (nextRank == Y軸 + movableRankVector[i]))
			return true;
	}
	return false;
}

/**
 * 駒が指定した座標に移動できるか
 * 他の駒も考慮して移動できるかを判断する
 * @param int[][] board 現在の盤
 * @param int nextFile 移動したいX座標
 * @param int nextRank 移動したいY座標
 * @return 移動できるか
 */
public boolean movable (int[][] board, int nextFile,int nextRank) {
	if (!isOnBoard) // すでに取られている場合
		return false;
	ArrayList<NextMove> movableList = movableList (board);
	for (int i = 0; i < movableList.size(); ++i) {
		if ((movableList.get(i).nextFile() == nextFile)
		&& (movableList.get(i).nextRank() == nextRank)) {
			return true;
		}
	}
	return false;
}

/**
 * 駒を指定した座標に移動する
 * 他の駒は無視して自分が移動できるかのみを判断し、可能なら移動する
 * @param int nextFile 移動したいX座標
```

```
 * @param int nextRank 移動したいY座標
 * @return 移動できたか
 */
public boolean checkAndMove (int nextFile,int nextRank) {
        if (movable (nextFile, nextRank)) {
                X軸 = nextFile;
                Y軸 = nextRank;
                return true;
        } else return false;
}

/**
 * 駒を指定した座標に移動する
 * 他の駒の位置も考慮して移動できるか判断し、可能なら移動する
 * @param int[][] board 現在の盤
 * @param int nextFile 移動したい x座標
 * @param int nextRank 移動したい y座標
 * @return 移動できたか
 */
public boolean checkAndMove (int[][] board, int nextFile,int nextRank) {
        if (movable (board, nextFile, nextRank)) {
                X軸 = nextFile;
                Y軸 = nextRank;
                return true;
        } else return false;
}

/**
 * 移動可能な座標のリストを返す
 * @param int[][] board 現在の盤
 * @return 移動可能な座標のリスト
 */
public ArrayList<NextMove> movableList (int[][] board) {
        ArrayList<NextMove> movableList = new ArrayList<NextMove>();
        boolean[][] isMovable = {{false, false, false, false, false}, // 移動可能な位置
                                 {false, true,  true,  true,  false},
                                 {false, true,  true,  true,  false},
                                 {false, true,  true,  true,  false},
                                 {false, true,  true,  true,  false},
```

```
                        {false, true,  true,  true,  false},
                        {false, false, false, false, false}};

switch (type) { // 他の駒による移動不可能な位置の判定
case ANPANMAN : // アンパンマンの場合
        for (int r = 1; r <= 5; ++r) {
                for (int f = 1; f <=3; ++f) {
                        switch (board[r][f]) {
                        case ANPANMAN :     // 自分の駒がある位置へは移動不可
                        case SHOKUPANMAN :
                        case CURRYPANMAN :
                                isMovable[r][f] = false;
                                break;
                        case BAIKINMAN :    // バイキンマンに取られる位置へは移動不可
                                isMovable[r][f-1] = false;
                                isMovable[r+1][f-1] = false;
                                isMovable[r+1][f] = false;
                                isMovable[r+1][f+1] = false;
                                isMovable[r][f+1] = false;
                                break;
                        case HORRORMAN :    // ホラーマンに取られる位置へは移動不可
                                isMovable[r][f-1] = false;
                                isMovable[r+1][f] = false;
                                isMovable[r][f+1] = false;
                                break;
                        case DOKINCHAN :    // ドキンちゃんに取られる位置へは移動不可
                                isMovable[r+1][f-1] = false;
                                isMovable[r+1][f] = false;
                                isMovable[r+1][f+1] = false;
                                break;
                        }
                }
        }
case SHOKUPANMAN : // 食パンマンの場合
case CURRYPANMAN : // カレーパンマンの場合
        for (int r = 1; r <= 5; ++r) {
                for (int f = 1; f <=3; ++f) {
                        switch (board[r][f]) {
                        case ANPANMAN :     // 自分の駒がある位置へは移動不可
```

```
                        case SHOKUPANMAN :
                        case CURRYPANMAN :
                                isMovable[r][f] = false;
                                break;
                        }
                }
        }
        break;
                case BAIKINMAN : // バイキンマンの場合
        for (int r = 1; r <= 5; ++r) {
                for (int f = 1; f <=3; ++f) {
                        switch (board[r][f]) {
                        case BAIKINMAN :      // 自分の駒がある位置へは移動不可
                        case HORRORMAN :
                        case DOKINCHAN :
                                isMovable[r][f] = false;
                                break;
                        case ANPANMAN :     // アンパンマンに取られる位置へは移動不可
                                isMovable[r][f-1] = false;
                                isMovable[r-1][f-1] = false;
                                isMovable[r-1][f] = false;
                                isMovable[r-1][f+1] = false;
                                isMovable[r][f+1] = false;
                                break;
                        case SHOKUPANMAN :     // 食パンマンに取られる位置へは移動不可
                                isMovable[r][f-1] = false;
                                isMovable[r-1][f] = false;
                                isMovable[r][f+1] = false;
                                break;
                        case CURRYPANMAN :     // カレーパンマンに取られる位置へは移動不可
                                isMovable[r-1][f-1] = false;
                                isMovable[r-1][f] = false;
                                isMovable[r-1][f+1] = false;
                                break;
                        }
                }
        }
case HORRORMAN : // ホラーマンの場合
case DOKINCHAN : // ドキンちゃんの場合
```

```
                    for (int r = 1; r <= 5; ++r) {
                            for (int f = 1; f <=3; ++f) {
                                    switch (board[r][f]) {
                                    case BAIKINMAN :      // 自分の駒がある位置へは移動不可
                                    case HORRORMAN :
                                    case DOKINCHAN :
                                            isMovable[r][f] = false;
                                            break;
                                    }
                            }
                    }
                    break;
            }
            for (int i = 0; i < movableFileVector.length; ++i) { // 移動可能かチェック
                    int nextFile = X軸 + movableFileVector[i];
                    int nextRank = Y軸 + movableRankVector[i];
                    if (isMovable [nextRank][nextFile]) {
                            NextMove nextMove = new NextMove (type, nextFile, nextRank); // 移動可能リス
トに挿入
                            movableList.add (nextMove);
                    }
            }
            return movableList;
    }

    /**
     * 移動可能な座標を表示する
     * @param int[][] board 現在の盤
     */
    public void showMovable (int[][] board) {
            ArrayList<NextMove> movableList = movableList (board); // 移動可能な座標の判定

            if (movableList.size() == 0) { // 駒が移動可能な位置が無い場合
                    System.out.println (name() + "が進める位置はありません");
            } else {
                    System.out.print (name() + "は現在(" + X軸 + "," + Y軸 + ")にいて");
                    for (int i = 0; i < movableList.size(); ++i) {
                            int nextFile = movableList.get(i).nextFile();
                            int nextRank = movableList.get(i).nextRank();
```

```
                    System.out.print ("(" +  nextFile + "," + nextRank + ")");
                }
                System.out.println ("へ移動できます");
            }
        }

        /**
         * クローン生成
         */
        public Piece clone() {
            Piece newPiece = new Piece (this.type, this.X軸, this.Y軸);
            if (!isOnBoard) newPiece.removeFromBoard();
            return newPiece;
        }
}
```