卒業研究報告書

題目

三次元チェスの対戦 AI 開発

指導教員

石水　隆　講師

報告者

11-1-037-0122

金谷　洋佑

近畿大学理工学部情報学科

平成 27 年 1 月 31 日提出

概要

チェスには様々なバリエーションが存在する。その一つに三次元チェスがある。三次元チェスは通常のチェスと違い駒を三次元的に動かすものを指す。三次元チェスは立体的なチェス盤が必要となるため簡単にはプレイすることができない。そのため、計算機上でプレイできる環境を構築する必要がある。一方、三次元チェスはまだ対戦できるAIがあまり多くない。そこで本研究では三次元チェスの一つであるラオムシャッハ(Raumschach)の対戦AIの開発を目指す。

一般的なチェスのAIでは、定跡データベースを利用して、序盤は定跡に従って指すようにしている。しかし、三次元チェスでは定跡が確立していない。そこで各駒に性能に応じた価値を付加し、先読みにより盤面評価値を求め、最も評価値の高い手を指す戦略を目指す。

本格的な三次元チェスのAI開発に先立ち本研究ではまずポーンのみからなる三次元チェスのプログラムをJavaを用いて作成した。

目次

1　序論

1.1.　本研究の背景

チェスには様々なバリエーションが存在する。チェスのルール、駒、盤の一部を変更したチェスは変則チェスと呼ばれる。その一つに三次元チェスがある。三次元チェスは通常のチェスとは違い駒を三次元的に動かして指すことができる。三次元チェスにも様々なバリエーションがあるが、その中でも最も古いものの一つとしてラオムシャッハ(Raumschach)がある。ラオムシャッハは 1907 年に Ferdinant Maack が発明したチェスであり、5x5x5 のチェス盤使用する。

ラオムシャッハ等の三次元チェスは立体的な盤が必要となるため、簡単にプレイすることができない。そのため、計算機上でプレイできる環境を構築する必要がある。一方、三次元チェスまだ対戦できる AI がおまり多くない。

既存の 3 次元チェス AI としてはとしては 3D CHESS[3]等がある。また、既存のラオムシャッハ AI には[4][5][6]等がある。[4]の AI は Random、Easy、Fast、Medium、Strong があり強さを選ぶことができる Random はランダムで移動できる駒を適当に移動させる Fast は Easy より弱いが 2 秒以内に駒を動かすように作ってある。後は Easy、Medium、Strong の順に強い。

1.1. ゲーム AI の手法

チェスのように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[7]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。

先読みとは、可能な範囲で一定数の先の手を読み、その手から作られる局面の評価値を求め、最も評価値が高い手を採用することである。一般に先読みする手数が多いほど強い AI となるが、先読み手数の増加に伴い探索時間が指数的に増えるため、適度に枝刈りをして探索範囲を減らす工夫をする必要がある。探索アルゴリズムにはミニマックス法、$\alpha-\beta$ 法と言う手法がある。基本的にはミニマックス法と $\alpha-\beta$ 法は同じであり、同じ答えが得られる。しかし、ミニマックス法は全ての手を計算するため時間がかかってしまう、$\alpha-\beta$ 法はしなくていい部分を計算しない分効率が良い。ミニマックス法，α - β 法は共に自分の評価値が最大になる手を探し、相手の評価値が最小になる手を探す手法である。

局面の評価値計算とは局面の優劣を数値化して計算を行うことを言い、先手有利な局面であれば正の値より先手有利なほど絶対値は大きくなる。後手有利な局面であれば負の値より後手有利なほど絶対値は大きくなる。

定跡データベースとは一般的には序盤が定跡化されており、指し手の選択によって、先手有利、後手有利などの変化が生じる。戦法によっては、終盤まで定跡化されていることもあり、序盤での消費時間やケアレスミスを回避するため多数の定跡をデータベースかすることを言う。

必勝読みとは勝敗を読みきることを言う、また完全読みとはどのように勝つかまで読みきることを言う。

モンテカルロ法とは乱数を用いたシミュレーションや数値計算を行うことにより近似解を求める計算手法である。

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

1.2. 本研究の目的

本研究では三次元チェスの一つであるラオムシャッハ(Raumschach)の対戦 AI の開発を目指す。一般的なチェスの AI では、定跡データベースを利用して序盤は定跡に従って指すようにしている。しかし、三次元チェスでは定跡が確立していない。そこで各駒に性能に応じた価値を付加し、先読みにより盤

面評価値を求め、最も評価値の高い手を指す戦略を目指す。
本格的な三次元チェスの AI 開発に先立ち本研究ではまずポーンのみからなる三次元チェスのプログラムを Java を用いて作成した。

1.3. 本報告書の構成

本報告書の構成は以下の通りである。二章ではラオムシャッハについての説明をする。三章では本研究で作成したプログラムについて述べる。

2. ラオムシャッハ(Raumschach)

1907 年にドイツのフェルディナント・マック (Ferdinand Maack) によって発明された。ラオムシャッハは変則チェスの一種であり、チェスを三次元に拡張したものである。ラオムシャッハは通常のチェスと同じく、二人のプレイヤーが互いのキングを取り合うゲームである。チェス盤のマスは三次元上に配置され、各駒が移動できる範囲も三次元に拡張されている。

2.1. ルール

ラオムシャッハのチェス盤は立方体を 5 × 5 × 5 に分けた 125 マスの空間で構成される。各階層 (Column) は下層より順に大文字の A~E で表し、通常のチェス同様列 (File) は小文字の a~e、行 (Rank) は数字の 1~ 5 で表す。各マスは Bb5 のように階層, 列, 行を順に並べて表記される。駒は通常のチェスの 6 種類 キング (King)・クイーン (Queen)・ビショップ (Bishop)・ナイト (Knight)・ルーク (Rook)・ポーン (Pawn)に加えて、ラオムシャッハ独自の駒ユニコーン (Unicorn) を合わせ 7 種類であり、白駒は階層 A,B の 1,2 行に、 黒駒は階層 D,E の 4,5 列に配置される。また、図 1 にラオムシャッハのチェス盤と駒の初期配置を示す。図 1 中の赤い文字は白駒の位置を、黒い文字は黒駒の位置を表す。
ゲームは白と黒に分かれた二人のプレイヤーにより行われる。先手は白でプレイヤーは交互に自分の駒を一 回ずつ動かす。パスをすることはできない。自分の手番の時に自分の駒の動ける範囲に敵の駒が存在するなら ば、敵の駒をチェスボードから取り除き自分の駒を敵の駒のあった場所に移動することができる。(例外:ポーン) 以下、これを攻撃と記述する相手のキングを攻撃できる状態にする手を「チェック (Check)」という。 いかなる場合においてもキングはチェックされる位置に移動することはできない。キングが絶対に逃げられな いように追い詰めたチェックを「チェックメイト (Checkmate)」と呼ぶ。双方は相手のキングをチェックメイトすることを目指す。

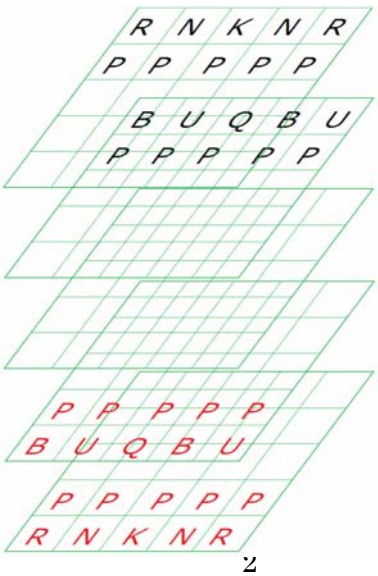

図１ ラオムシャッハの初期配置

2.2. 駒の動き

ラオムシャッハの駒の動き方は、通常のチェスの駒の動きを三次元方向に拡張したものとなっている。以下では（x, y, z）（a ≤ x ≤ e, 1 ≤ y ≤ 5, A ≤ z ≤ E）をラオムシャッハのチェス盤の（列, 行, 階層）座標とする。

- **キング**
  キングはすべての方向に 1 マス移動できる。キングが（x, y, z）にいる場合、(x±1, y±1, z±1), (x, y±1, z±1), (x±1, y, z±1), (x, y±1, z), (x±1, y, z)のいずれか一ヶ所に移動できる。
- **クイーン**
  クイーンはビショップ、ルーク、ユニコーンのすべての動きを兼ね揃えた駒である。クイーンは 27 個のベクトル(x, y, z)(x, y, z = {-1, 0, 1})のうち 0 ベクトル(0, 0, 0)を除く 26 個のベクトルのいずれか 16 方向へ他の駒または終端に当たるまで移動できる。
- **ナイト**
  ナイトは各面に向かってニマス進み、そこから左右に一マス進んだ計 24 ヶ所に移動できる。初期位置(x, y ,z)にいるナイトは (x±2, y±1, z), (x±2, y, z±1), (x±1, y±2, z), (x, y±2, z±1), (x±1, y, z±2), (x, y±1, z±2)のいずれかの場所に移動できる。
- **ビショップ**
  ビショップを立方体の中心部においた場合、各面の中央に向かう 12 方向に移動することができる。ビショップは初期位置からベクトル(0, ±1, ±1), (±1, ±1, 0), (±1, 0, ±1)のいずれかの一つの方向へ他の駒または終端に当たるまで移動できる。
- **ルーク**
  ルークを立方体の中心部に置いた場合、各面の中央に向かう 6 方向に移動できる。ルークは初期位置からベクトル(0, 0, ±1), (0, ±1, 0), (±1, 0, 0)のいずれか一つの方向へ他の駒または終端に当たるまで移動できる。
- **ユニコーン**
  ラオムシャッハ独自の駒であるユニコーンは立方体の頂点方向八方向に可能な限り動くことができる。ユニコーンは初期位置からベクトル(±1, ±1, ±1)のいずれか一つの方向へ他の駒または終端に当たるまで移動できる。

図 2、図 3、図 4、図 5、図 6、図 7 に各駒の動き方を示す。灰色の円が各駒を Cc3 に置いたときにその駒の移動できる範囲を示している。5 枚の盤面は三次元チェスの盤面を 2 次元に表したものであり、5 枚の盤面は上からＡ~Ｅの階層である。

- **ポーン**
  ポーンは移動できるマスと攻撃できるマスが異なる。移動する場合、ポーンは前へ 1 マス、もしくは相手陣地に進む方向へ階層を一層移動することができる。ポーンの攻撃できるマスは 1 マス進んだ左右のマスである。ポーンが攻撃した場合も他の駒の攻撃と同様に敵の駒のあった場所にポーンを移動させる。つまり初期位置(x, y, z)にいる白駒のポーンは(x, y＋1, z), (x, y, z＋1)のどちらかに移動でき、(x±1, y＋1, z), (x±1, y, z＋1), (x, y＋1, z＋1)のいずれかに黒駒があればその駒を取ってその位置へ移動できる。黒駒のポーンは(x, y－1, z), (x, y, z－1)のどちらかも移動でき、(x±1, y－1, z), (x±1, y, z-1)(x, y－1, z－1)のいずれかに白駒があればその駒を取ってその位置に移動できる。また、通常のチェスでは初期位置から動いていないポーンは 2 マス移動する 2 段跳ねを行うことができるが、ラオムシャッハでは 2 段跳ねはできない。

図 8、図 9 にポーンの動きを示す。灰色の円がその駒の移動できる範囲を示していて、赤色の円は敵の駒がその位置にいないと動けない。5 枚の盤面は三次元チェスの盤面を 2 次元に表したものであり、5 枚の盤面は上からA~Eの階層である。

2.3. その他のルール

ラオムシャッハにはステールメートとプロモーションというルールがある

- **ステールメート(Stalemate)**
  キングはチェックされる位置に移動することはできない。よって残りの駒が極端に少なくなった場合、どの駒も動かすことができない状態が発生することがある。これをステールメートと言い、ゲームは引き分けとなる。
- **プロモーション(Promotion)**
  ポーンがそれ以上進めない位置に到達したとき、そのポーンをキング以外の別の駒に変化させることができる。これをプロモーションまたは昇格と言う。
- **キングのルール**

  キングは敵駒の利いているマスへは移動で利きない。駒を動かした結果、自キングに敵駒の利きが入る手はさせない。自キングにチェックがかかっている場合はチェックから逃れる手以外はさせない。

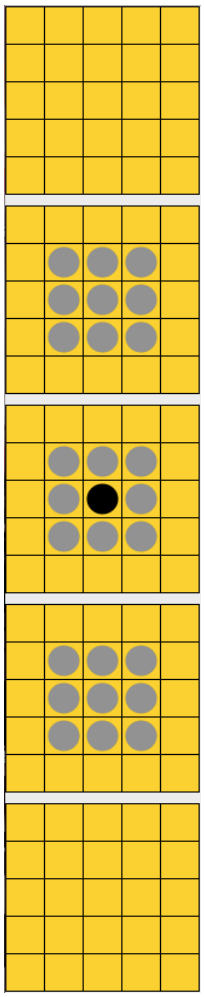

図２　キングの動き

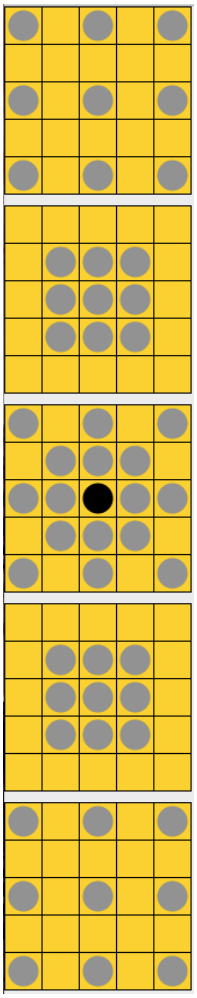

図３　クイーンの動き

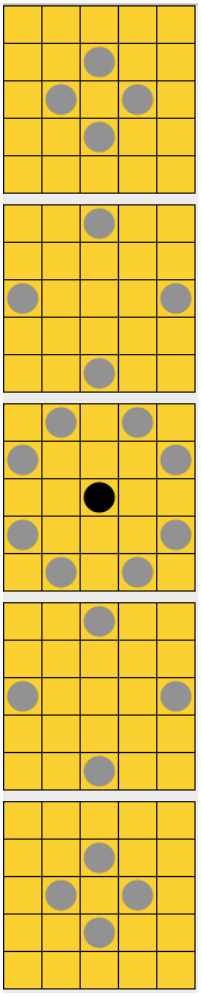

図４　ナイトの動き

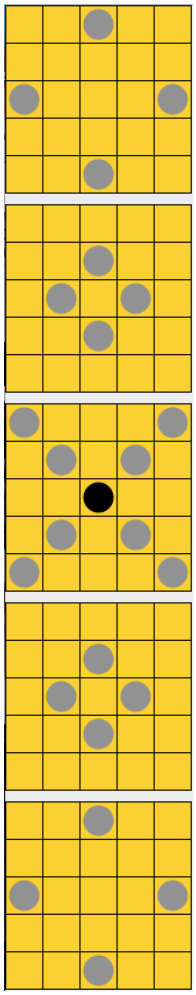

図 5　ビショップの動き

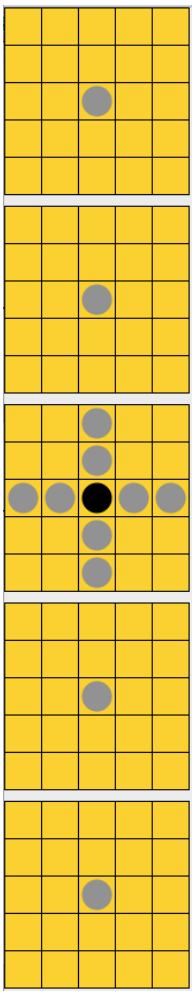

図 6　ルークの動き

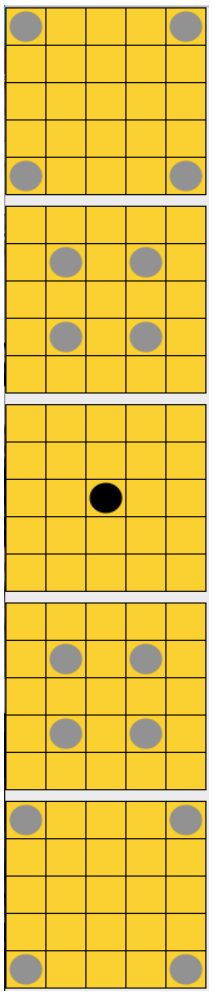

図 7　ユニコーンの動き

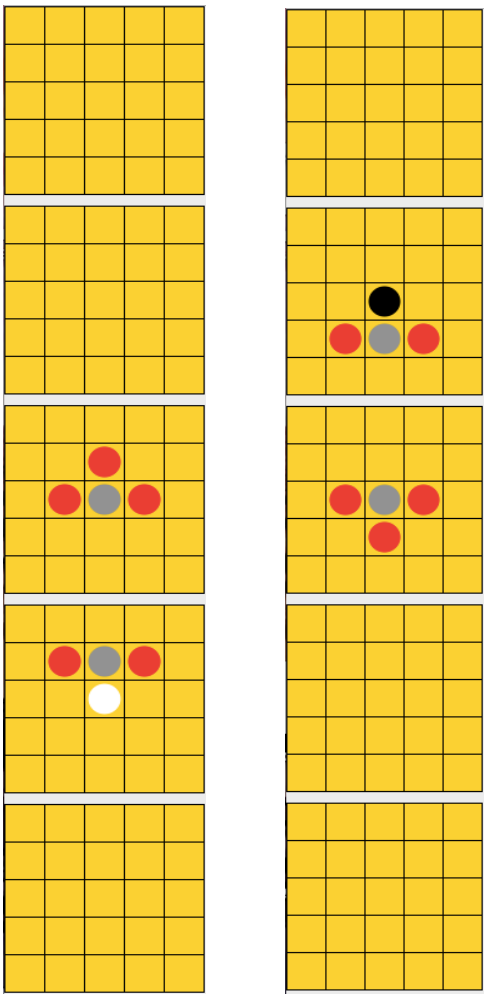

図 8　白ポーンの動き　図 9　黒ポーンの動き

3.　ポーンのみからなる三次元チェス

ラオムシャッハプログラムの開発に先立ち、本研究ではポーンのみの三次元チェスプログラムを作成した。

3.1. 簡易版三次元チェスのルール

簡易版三次元チェスの駒はポーンの動きをする駒のみである。ポーンの駒は三次元チェスと同じ位置であり個数もある同じである。自駒のどれか一つでもそれ以上進めない位置に到達したときに勝利である。

4.　本研究で作成したプログラム

本研究では、簡易版三次元チェスプログラムを作成した。付録 1 に本研究で作成した簡易版チェスプログラムのソースを示す。

本研究で作成したプログラムでは、手番プレイヤーは各自の駒を、ラオムシャッハのポーンの動きで移動させることができる。マウスクリックのみで駒を選択・移動することができる。自駒をクリックするとその駒が選択され、その駒が移動するできるマスを表示される。続けて移動したいマスをクリックすると移動できる。

Chess.java と MainPanel.java の２つのクラスからなるプログラムである。

- **Chess.java**
  Chess.java はゲームを起動するためのメインクラスであり、タイトルやサイズの固定などをなどもこのクラスでやっている。
  - Chess
    タイトルの設定とサイズの固定などをする
    setTitle()：このメソッドでフレームにタイトルをつけることできる
    setResizable()：サイズ変更の可否を設定することができる
  - main
    ゲームを起動する

- **MainPanel.java**
  MainPanel.java はゲーム本体の中身が書いてあるクラスである。

  - MainPanel

    盤面の初期化とマウス操作を受け付けるようにする。
    setPreferredSize()：サイズの設定をすることができる
    addMouseListener(this)：マウス操作を受け付けるようにできる

  - initBoard

    駒の初期配置をする

```
for(int x = 0; x < MASU; x++) {
        for(int y = 0; y < MASU; y++ ) {
                for(int z = 0; z < MASU; z++){
                        board[x][y][z] = BLANK;
                }
        }
}
```

  - drawBoard

    盤面の描画をする
    g.setColer()：色をつけることができる、線に色をつけている
    g.drawLine()：線を引くことができる、盤面を描画するために縦線、横線を引いている

  - imagePiece

    駒の描画をする
    g.fillOval()：円を描くことができる、駒はポーンしかないため色を変えて円で駒を表している

  - drawString

    文字の描画をする
    g.darwString()：文字を描画することができる、盤面の横に階層を文字として書いている

  - endGame

    勝敗を表示し盤面を初期化する、initBoard()を使って盤面を初期化している

- moveCheck

  駒が移動できるかをチェックする

```
If(board[x][y+1][z] == BLANK){
        board[x][y+1][z] = BPM;
 }
 If(board[x-1][y+1][z] < 0){
        board[x-1][y+1][z] = BL;
 }
```

  駒の移動するところが BLANK だったらその位置に移動して灰色の円を描画する。攻撃できる位置に敵駒があったら尾回路の円を描画する。

- PieceMove

  駒を移動させる。

```
If(board[x][y][z] == BPM || board[x][y][z] == BL){
        board[x][y][z] = BP;
}
```

  灰色の円か赤色の円をクリックすることで駒を移動することができるようしている。

5. 結果および考察

プログラムの実行結果を図 10 で示す。

[4]と同じで盤面を二次元に作っているが[4]とは違い縦一列に盤面を置いてあり[4]より駒の移動位置がわかりやすい、[5][6]の盤面は三次元で作られているため二次元で作っているよりかは盤面が見やすい。[4]は駒をクリックするともう一度同じ駒をクリックしないと駒の移動位置表示をキャンセルすることができないが、別の駒をもしくは空白をクリックすることで移動位置表示をキャンセルできるため操作性は[4]よりは操作しやすい。

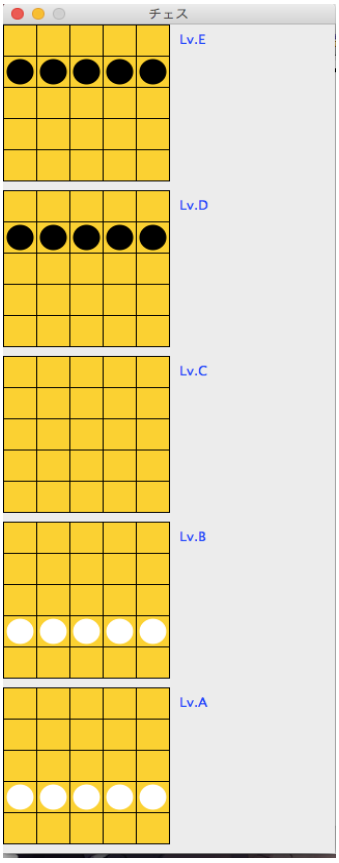

図 10　実行結果

6.　結論・今後の課題

本研究では三次元チェスの対戦 AI 開発に先立ち、ポ ーンのみからなる簡易版チェスの作成をした。

本研究で作成したプログラムは、ラオムシャッハのポーンの動きに従って駒を動かすことができる。ポーン以外の駒の動きもできるようにして、ラオムシャッハとしてプレイできるようにすること、駒を増やした時の分かりやすさの向上させることが今後の課題である。また、対戦 AI の作成も今後の課題である。作成する AI は、先読みにより盤面の評価値を求め、最も評価値 の高い手を指すものを目指す。先読みを行う際に幅優先 探索では時間かかるが完全性があり、深さ優先探索では幅優先探索よりは時間はかからないが不完全になる場合がある。その両方のいいところを併せ持つ反復深化深さ優先探索使い今後対戦 AI を開発を目指す。

謝辞

本報告書の制作にあたり、数え切れないほどのご指導、ご助力など大変尽力していただき、石水隆講師には誠に感謝申し上げます。本当にお世話になりました。

参考文献

[1] A.S.M.Dickins, “Guide to Fairy Chess,” Dover Publications Inc, (1971).

[2] 松田道弘:世界のゲーム辞典 pp204,205,東京堂出版 (1989).

[3] Dan Beyer, 3D CHESS, (2006). http://thehinge.net/3dchess/

[4] Jcfrog, Three-dimensional Chess: Raumschach , (2014), https://fr.jocly.com/raumschach

[5] L. Lynn Smith, , Game: Emperor Raumschach, (2005), http://www.zillions-of-games.com/cgi-bin/zilligames/submissions.cgi?do=show;id=1116

[6] Robert Price, Game: Raumschach , (2000), http://www.zillions-of-games.com/cgi-bin/zilligames/submissions.cgi?do=show;id=708

[7] 美添一樹, 山下宏, 松原仁, コンピュータ囲碁―モンテカルロ法の理論と実践―, 共立出版, (2012)

付録１　ポーンのみからなる三次元チェスのソースコード

Chess.java

```
import java.awt.*;
import javax.swing.*;

public class Chess extends JFrame {
	public Chess() {
		// タイトルを設定
		setTitle("チェス");
		// サイズ変更をできなくする
		setResizable(false);
		// メインパネルを作成してフレームに追加
		MainPanel panel = new MainPanel();
		Container contentPane = getContentPane();
		contentPane.add(panel);

		// パネルサイズに合わせてフレームサイズを自動設定
		pack();
	}

	public static void main(String[] args) {
		Chess frame = new Chess();
		frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		frame.setVisible(true);
	}
}
```

MainPanel.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class MainPanel extends JPanel implements MouseListener{
		//マスのサイズ
		private static final int GS = 32;
		//マスの数　３次元チェスは 5×5×5 マス
		private static final int MASU = 5;

		//盤面の大きさ
		private static final int WIDTH = GS * MASU;
		private static final int HEIGHT = WIDTH;
```

```
	//盤面
	private int[][][] board = new int[MASU][MASU][MASU];
	private int[][][] tmp = new int[MASU][MASU][MASU];
	// 空白
	private static final int BLANK = 0;

	//白の番
	private boolean flagForWhite;

	//黒駒
	private static final int BP = 1; // ポーン
	//白駒
	private static final int WP = -1;

	//白、黒の駒チェック用
	private static final int BPM =  11;

	private static final int WPM = -11;

	//取れるか取れないか
	private static final int BL =  21;

	private static final int WL =  -21;

	JLabel label;

	Image iwp;
	Image ibp;

	private static int xc;
	private static int yc;
	private static int zc;
	private static int ch = 1;;


	public MainPanel() {
		//Chess で pack()するときに必要
		setPreferredSize(new Dimension(WIDTH*2, HEIGHT * MASU + 50));
		//盤面を初期化する
		initBoard();

		//画像を登録
		Iwp
=Toolkit.getDefaultToolkit().getImage(getClass().getResource("WP.png"));
		iwp
=Toolkit.getDefaultToolkit().getImage(getClass().getResource("BP.png"));
```

```
		//マウス操作を受け付けるようにする
		addMouseListener(this);


	}

	public void paintComponent(Graphics g) {
		//super.printComponent(g);

		//盤面を書く
		drawBoard(g);
		//文字を書く
		drawString(g);
		//駒を描画する
		imagePiece(g);

	}


	private void initBoard() {


		for(int x = 0; x < MASU; x++) {
			for(int y = 0; y < MASU; y++ ) {
				for(int z = 0; z < MASU; z++){
					board[x][y][z] = BLANK;
				}
			}
		}




		//初期配置
		//黒ポーン
		board[0][1][0] = board[1][1][0] = board[2][1][0] = board[3][1][0] = board[4][1][0] = board[0][1][1] = board[1][1][1] = board[2][1][1] = board[3][1][1] = board[4][1][1] =BP;


		//白ポーン
		board[0][3][3] = board[1][3][3] = board[2][3][3] = board[3][3][3] = board[4][3][3] = board[0][3][4] = board[1][3][4] = board[2][3][4] = board[3][3][4] = board[4][3][4] = WP;


		for(int x = 0; x < MASU; x++) {
			for(int y = 0; y < MASU; y++ ) {
```

```
                        for(int z = 0; z < MASU; z++){
                            tmp[x][y][z] = board[x][y][z];
                        }
                    }
                }


                flagForWhite = true;

        }

        private void drawBoard(Graphics g){
                g.setColor(Color.orange);
                for(int x = 0; x < MASU; x++){
                  g.fillRect(0, 170*x, WIDTH , HEIGHT);

                }


                //マス枠を描画
                g.setColor(Color.black);

                //縦線を引く
                for(int i = 1; i < MASU; i++){
                        for(int l = 1; l < MASU; l++){
                        g.drawLine(i*GS, 1, i*GS , HEIGHT);
                        g.drawLine(i*GS, 170*l, i*GS , HEIGHT+170*l);
                        }
                }
                //横線を引く
                for(int i = 1; i < MASU; i++){
                        for(int l = 1; l < MASU; l++){
                            g.drawLine(0, i*GS, WIDTH, i*GS);
                            g.drawLine(0, i*GS+170*l, WIDTH, i*GS+170*l);
                        }
                }
                // 外枠
                for(int i = 0; i < MASU; i++){
                        g.drawRect(0,170*i,WIDTH, HEIGHT);
                }

        }

        private void imagePiece(Graphics g){

        for (int x = 0; x < MASU; x++) {
            for (int y = 0; y < MASU; y++) {
                for(int z = 0; z < MASU; z++){
```

```
                    if (board[x][y][z] == BLANK) {
                            continue;
                    } else if (board[x][y][z] == BP) {
                            g.setColor(Color.BLACK);
                    } else if (board[x][y][z] == WP) {
                            g.setColor(Color.WHITE);
                    } else if (board[x][y][z] == BPM) {
                            g.setColor(Color.GRAY);
                    } else if (board[x][y][z] == WPM) {
                            g.setColor(Color.GRAY);
                    } else if (board[x][y][z] == BL) {
                            g.setColor(Color.RED);
                    } else if (board[x][y][z] == WL) {
                            g.setColor(Color.RED);
                    }else {

                    }
                    g.fillOval(x * GS + 3, y * GS + 3 + (170 * z),GS - 6, GS - 6);
            }
        }
}
}



private void drawString(Graphics g){


            g.setColor(Color.blue);
            g.drawString("Lv.E", GS*MASU + 10   , 20 );
            g.drawString("Lv.D", GS*MASU + 10   , 30 + GS*MASU );
            g.drawString("Lv.C", GS*MASU + 10   , 40 + GS*MASU * 2);
            g.drawString("Lv.B", GS*MASU + 10   , 50 + GS*MASU * 3);
            g.drawString("Lv.A", GS*MASU + 10   , 60 + GS*MASU * 4);

}

private void endGame(){
            for(int x=0; x < MASU; x++){
                    if(board[x][4][4] == BP){
                            System.out.println("BLACK WIN");
                            initBoard();
                    }

                    if(board[x][0][0] == WP){
                            System.out.println("WHITE WIN");
                            initBoard();
                    }
```

```
		}

}



private void moveCheck(int x,int y,int z){

		ch = 0;
		if(!flagForWhite){
			if(board[x][y][z] == BP){				//ポーンの動き
				if(!(y==4)){
					if(board[x][y+1][z] == BLANK){
						board[x][y+1][z] = BPM;
					}
				}
				if(!(z==4)){
					if(board[x][y][z+1] == BLANK){
						board[x][y][z+1] = BPM;
					}
				}

				//駒を取る
				if(!(x==4) && !(y==4)){
					if(board[x+1][y+1][z] < 0){
						board[x+1][y+1][z] = BL;
					}
				}

				if(!(x==0) && !(y==4)){
					if(board[x-1][y+1][z] < 0){
						board[x-1][y+1][z] = BL;
					}
				}

				if(!(y==4) && !(z==4)){
					if(board[x][y+1][z+1] < 0){
						board[x][y+1][z+1] = BL;
					}
				}

				if(!(x==4) && !(z==4)){
					if(board[x+1][y][z+1] < 0){
						board[x+1][y][z+1] = BL;
					}
				}

				if(!(x==0) && !(z==4)){
```

```
                        if(board[x-1][y][z+1] < 0){
                                board[x-1][y][z+1] = BL;
                        }
                }
        }
}

else if(flagForWhite){
        if(board[x][y][z] == WP){  //ポーンの動き

                if(!(y == 0)){
                        if(board[x][y-1][z] == BLANK){
                                board[x][y-1][z] = WPM;
                        }
                }

                if(!(z == 0)){
                        if(board[x][y][z-1] == BLANK ){
                                board[x][y][z-1] = WPM;
                        }
                }

                //駒を取る
                if(!(x==4) && !(y==0)){
                        if(board[x+1][y-1][z] > 0){
                                board[x+1][y-1][z] = WL;
                        }
                }

                if(!(x==0) && !(y==0)){
                        if(board[x-1][y-1][z] > 0){
                                board[x-1][y-1][z] = WL;
                        }
                }

                if(!(y==4) && !(z==0)){
                        if(board[x][y-1][z-1] > 0){
                                board[x][y-1][z-1] = WL;
                        }
                }

                if(!(x==4) && !(z==0)){
                        if(board[x+1][y][z-1] > 0){
                                board[x+1][y][z-1] = WL;
                        }
                }

                if(!(x==0) && !(z==0)){
```

```
                                        if(board[x-1][y][z-1] > 0){
                                                board[x-1][y][z-1] = WL;
                                        }
                                }

                        }
                }
        }

        private void PieceMove(int x,int y,int z){

                if(board[x][y][z] > 10){
                        ch = 1;
                        board[xc][yc][zc] = BLANK;
                        if(board[x][y][z] == BPM || board[x][y][z] == BL){
                                board[x][y][z]    = BP;
                        }


                        for(int i = 0; i < MASU; i++) {
                                for(int l = 0; l < MASU; l++ ) {
                                        for(int m = 0; m < MASU; m++){

                                                if(board[i][l][m] > 20){
                                                        board[i][l][m] = tmp[i][l][m];
                                                }else if(board[i][l][m] > 10){
                                                        board[i][l][m] = BLANK;
                                                }

                                        }
                                }
                        }

                        for(int a = 0; a < MASU; a++) {
                                for(int b = 0; b < MASU; b++ ) {
                                        for(int c = 0; c < MASU; c++){
                                                tmp[a][b][c] = board[a][b][c];
                                        }
                                }
                        }



                        flagForWhite = !flagForWhite;
                }


                if(board[x][y][z] < -10){
```

```
		ch = 1;
		board[xc][yc][zc] = BLANK;
		if(board[x][y][z] == WPM || board[x][y][z] == WL){
			board[x][y][z]  = WP;
		}
		for(int i = 0; i < MASU; i++) {
			for(int l = 0; l < MASU; l++ ) {
				for(int m = 0; m < MASU; m++){
					if(board[i][l][m] < -20){
						board[i][l][m] = tmp[i][l][m];
					}else if(board[i][l][m] < -10){
						board[i][l][m] = BLANK;
					}

				}
			}
		}

		for(int a = 0; a < MASU; a++) {
			for(int b = 0; b < MASU; b++ ) {
				for(int c = 0; c < MASU; c++){
					tmp[a][b][c] = board[a][b][c];
				}
			}
		}

		flagForWhite = !flagForWhite;
	}



	if(board[x][y][z] == BLANK){		//空白を押したら選択をキャンセルする

		ch = 1;

		for(int a = 0; a < MASU; a++) {
			for(int b = 0; b < MASU; b++ ) {
				for(int c = 0; c < MASU; c++){
					board[a][b][c] = tmp[a][b][c];
				}
			}
		}

	}

	if(board[x][y][z] == BP || board[x][y][z] == WP){
		ch = 1;
```

```
                    for(int a = 0; a < MASU; a++) {
                            for(int b = 0; b < MASU; b++ ) {
                                    for(int c = 0; c < MASU; c++){
                                            board[a][b][c] = tmp[a][b][c];
                                    }
                            }
                    }
            }
}




@Override
public void mouseClicked(MouseEvent e) {
            //   どこのマスかを調べる
            int x = e.getX() / GS;
            int y = e.getY() / (GS+2);

      int z = y / 5 ;

            y = y % 5;


            //駒を移動させる
            PieceMove(x,y,z);

            if(ch == 1){
                    xc = x;
                    yc = y;
                    zc = z;
            }


            //駒の移動先を表示
            moveCheck(x,y,z);

            endGame();




            //System.out.println("x:" + x + ",y:" + y +",z:" + z );
            System.out.println("x:" + xc + ",y:" + yc +",z:" + zc );
            //label.setText("x:" + x + ",y" + y);
```

```
                //盤面の変化を再描画する
                //repaint();
                update(getGraphics());
        }
        public void mouseReleased(MouseEvent e) {

        }


        public void mouseEntered(MouseEvent e) {


        }


        public void mouseExited(MouseEvent e) {
                //

        }


        public void mousePressed(MouseEvent e) {

        }
}
```