Tic Tac Toe (CUI - java)

S.Matoike

目次

第 1 章

java による三目並べ [Tic Tac Toe]

1.1　制作する三目並べの概要

プログラムを実行すると、盤面が表示され、×の石を置く場所を指定するよう促されます

画面上に示された番号を入力すると、その番号のスロットに×の石が置かれた盤面が表示され、次の手番の○に、石を置く場所を指定するように促されます

手番を交互に変えながらゲームは進み、縦、横、斜めの何れかに、先に一列に自分の石を並べた方が勝ちとなります

既に石の置かれているスロット番号を指定できませんし、スロット番号として 0 〜 8 以外の数値を指定することもできません

```
スタート！ [Tic Tac Toe]
/---|---|---\
| 0 | 1 | 2 |
|---|---|---|
| 3 | 4 | 5 |
|---|---|---|
| 6 | 7 | 8 |
\---|---|---/
'X' さんの turn です
石を置く場所 0 〜 8 を指定して下さい : 4
/---|---|---\
| 0 | 1 | 2 |
|---|---|---|
| 3 | X | 5 |
|---|---|---|
| 6 | 7 | 8 |
\---|---|---/
'O' さんの turn です
石を置く場所 0 〜 8 を指定して下さい : 2
/---|---|---\
| 0 | 1 | O |
|---|---|---|
| 3 | X | 5 |
|---|---|---|
| 6 | 7 | 8 |
\---|---|---/
'X' さんの turn です
石を置く場所 0 〜 8 を指定して下さい :
```

1.2　定数定義のクラスと主処理

Constants クラスに定数の一式をおいて、他のクラスと共有するようにします

ソースコード 1.1　定数定義：Tic Tac Toe

```
1  package tictactoe;
2
3  public final class Constants {
4    private Constants(){}
5
6    final static public int NXN = 3;
7    final static public int WHITE = 10;
8    final static public int BLACK = -10;
9    final static public int MARU = WHITE;
10   final static public int BATSU = BLACK;
11   final static public int MAX = WHITE;
12   final static public int MIN = BLACK;
13   final static public int EMPTY = Integer.MIN_VALUE;
14   final static public int NEXT = 200;
15   final static public int DRAW = 100;
16   final static public int PROMPT = 1000;
17   final static public int RANDOM = 1010;
18   final static public int MINIMAX = 1001;
19   final static public int ALPHABETA = 1002;
20   final static public int MONTECARLO = 1003;
21 }
```

コマンドライン引数の指定で、先手と後手を入れ替えられるようにしています

ソースコード 1.2　main 関数：Tic Tac Toe

```
1  package tictactoe;
2  import static tictactoe.Constants.*;
3
4  public class TicTacToe {
5      public static void main(String[] args) {
6          boolean reverse = false;
7          if (args.length > 0) {
8              if (args[0].equals("-r")) {
9                  reverse = true;
10             }
11         }
12         Game g = new Game(reverse);
13         g.Start();
14     }
15 }
```

1.3　ゲームクラス

コンストラクタで、２人のプレイヤー、１つの盤面の各インスタンスを用意します

ソースコード 1.3　Game クラス：Tic Tac Toe

```
1  package tictactoe;
2  import static tictactoe.Constants.*;
3
4  public class Game {
5
6      private static Player[] player = null;
7      private static Board currBoard = null;
8      private static int current_player=0;
9
10     public Game(boolean reverse) {
11         player = new Player[2];
12         if (reverse) {
13             player[0] = new Player(MARU, "PC", MINIMAX);
14             player[1] = new Player(BATSU, "You", PROMPT);
15         } else {
16             player[0] = new Player(BATSU, "You", PROMPT);
17             player[1] = new Player(MARU, "PC", MINIMAX);
18         }
19         currBoard = new Board();
20     }
21
22     void Start() {
23         current_player = 0;
24         int winner = NEXT;
25         System.out.println("スタート！ [Tic Tac Toe]");
26         do {
27             currBoard.print();
28             Player currPlayer = player[current_player];
29             currPlayer.putStone(currBoard);
30             winner = currBoard.check();
31             current_player = ++current_player % 2;
32         } while (winner == NEXT);
33         currBoard.print();
34         result(winner);
35     }
36
37     private void result(int winner) {
38         // 結果を表示する関数
39     }
40
41 }
```

①盤面を表示し、②現在手番のプレーヤーに打つ手を決めさせ、③それを盤面に置きます　その次に④盤面の状態から勝敗を判定し、⑤手番を交代すること、これを勝敗がつくまで繰り返しています

勝敗の確認、board.check() の結果を引数として受け取り、それに基づいて結果を表示します
（上記のプログラム中で、//結果を表示する関数、の部分に収まる関数です）

ソースコード 1.4　結果を表示する関数：Tic Tac Toe

```
1  private void result(int winner) {
2      System.out.println("");
3      switch (winner) {
4        case DRAW:  System.out.print("引き分け\t");     break;
```

```
5         case MARU:  System.out.print("'O' の勝ち\t");  break;
6         case BATSU: System.out.print("'X' の勝ち\t");  break;
7       }
8       System.out.println("またね！");
9   }
```

1.4　石のクラス

石のクラスが持っているのは、その石が置かれる盤面上の位置（locate）と、そこへ置く石の種類（id=MARU or BATSU）です

private な変数には、public な getter と setter を通してアクセスさせます

ソースコード 1.5　石クラス：Tic Tac Toe

```
1   package tictactoe;
2   import static tictactoe.Constants.*;
3
4   public class Stone {
5       private int locate = 0;
6       private int color = BATSU;
7
8       Stone(int n, int i) {
9           locate = n;
10          color = i;
11      }
12      int getColor() {
13          return color;
14      }
15      void setColor(int i) {
16          color = i;
17      }
18      int getLocate() {
19          return locate;
20      }
21  }
```

1.5　盤面のクラス

盤面クラスが持っているのは、9 つの石を置く場所を持つゲーム盤 stones[0] 〜 stones[8] です

盤面の初期状態は、石を置く場所の全てが EMPTY の状態で、コンストラクタがその設定を行います

canPut(Stone) は、引数で受け取った石が、現在の盤面に置けるのかどうかをチェックします

setBoard(Stone) は、引数で受け取った石を、盤面上の指定された位置に置きます

ソースコード 1.6　盤面クラス：Tic Tac Toe

```
1   package tictactoe;
2   import static tictactoe.Constants.*;
3
4   public class Board {
5       private Stone[] stones = new Stone[NXN*NXN];;
6       public Board() {
```

```
7           for (int i = 0; i < NXN * NXN; i++) {
8               stones[i] = new Stone(i, EMPTY);
9           }
10      }
11      void setEmpty(int i){
12          stones[i]=new Stone(i, EMPTY);
13      }
14      void setBoard(Stone s) {
15          stones[s.getLocate()] = s;
16      }
17      Stone getBoard(int i){
18          return stones[i];
19      }
20      boolean canPut(Stone s) {
21          int n=s.getLocate();
22          if( !((0<=n)&&(n<NXN*NXN)) )  return false;
23          if (stones[s.getLocate()].getColor() != EMPTY)  return false;
24          return true;
25      }
26      void print() {  //盤面を表示する関数
27      }
28      private int lineSum(int n1, int n2, int n3) {
29          return stones[n1].getColor() + stones[n2].getColor() + stones[n3].getColor
                ();
30      }
31      int check() {   //勝敗を判定する関数
32      }
33      private boolean line(int n1, int n2, int n3){
34          return (( stones[n1].getColor()!=EMPTY ) &&
35                  ( stones[n1].getColor()==stones[n2].getColor() ) &&
36                  ( stones[n1].getColor()==stones[n3].getColor() ));
37      }
38      boolean isWin(){    // もう一つの勝敗を判定する関数
39      }
40      boolean isDraw(){   // 勝敗がついていないのに、
            EMPTYのスロットがもうないならDRAW
41      }
42      int evaluate(int turn){ // 局面を評価する（MINIMAX法が使う）
43      }
44  }
```

盤面上の石を置く場所には '0' 〜 '8' の数字を書き、MARU や BATSU の石の置かれた場所には、'O' あるいは 'X' の記号を書いています

（以下は、//盤面を表示する関数、の部分に収まる関数です）

ソースコード 1.7　盤面を表示する関数：Tic Tac Toe

```
1   public void print() {
2       char[] bd = new char[NXN*NXN];
3       for (int i = 0; i < NXN*NXN; i++) {
4           if (stones[i].getId() == MARU) {
5               bd[i] = 'O';
6           } else if (stones[i].getColor() == BATSU) {
7               bd[i] = 'X';
8           } else {
9               bd[i] = (char) ('0' + i);
```

```
        }
    }
    String fmt;
    System.out.println("\n/---|---|---\\");
    fmt = String.format("| %c | %c | %c |", bd[0], bd[1], bd[2]);
    System.out.println(fmt);
    System.out.println("|---|---|---|");
    fmt = String.format("| %c | %c | %c |", bd[3], bd[4], bd[5]);
    System.out.println(fmt);
    System.out.println("|---|---|---|");
    fmt = String.format("| %c | %c | %c |", bd[6], bd[7], bd[8]);
    System.out.println(fmt);
    System.out.println("\\---|---|---/");
}
```

勝敗を判定するには、横 3 行、縦 3 列、あるいは 2 つの斜めのライン上に、MARU あるいは BATSU が並んでいるかどうかを調べます

盤面上に EMPTY の状態の場所があったら、まだゲームは進行中（NEXT）です

盤面上に EMPTY の状態の場所が無くなっており、かつ勝敗がついていないのなら、それは引き分け（DRAW）です

（以下は、//勝敗を判定する関数、の部分に収まる関数です）

ソースコード 1.8　勝敗を判定する関数：Tic Tac Toe

```
public int check() {
    int i, l = 0;
    for (i = 0; i < (NXN+NXN+2); i++) {
        switch (i) {
            case 0: l = lineSum(0, 1, 2); break;
            case 1: l = lineSum(3, 4, 5); break;
            case 2: l = lineSum(6, 7, 8); break;
            case 3: l = lineSum(0, 3, 6); break;
            case 4: l = lineSum(1, 4, 7); break;
            case 5: l = lineSum(2, 5, 8); break;
            case 6: l = lineSum(0, 4, 8); break;
            case 7: l = lineSum(2, 4, 6); break;
        }
        if (l == NXN * MARU) {
            return MARU;
        } else if (l == NXN * BATSU) {
            return BATSU;
        }
    }
    for (i = 0; i < NXN*NXN; i++) {
        if (stones[i].getColor() == EMPTY) {
            return NEXT;
        }
    }
    return DRAW;
}
```

（以下は、//もう一つの勝敗を判定する関数、の部分に収まる関数です）

ソースコード 1.9　勝敗を判定する関数その 2 ：Tic Tac Toe

```
boolean isWin(){
    return (line(0,1,2)||
            line(3,4,5)||
            line(6,7,8)||
            line(0,3,6)||
            line(1,4,7)||
            line(2,5,8)||
            line(0,4,8)||
            line(2,4,6));
}
boolean isDraw(){
    int nEmpty=0;
    for(int i=0; i<NXN*NXN; i++){
        if(stones[i].getColor()==EMPTY)nEmpty++;
    }
    if(!isWin() && nEmpty==0)return true;
    return false;
}
```

（以下は、ゲームの局面を評価する関数で、後述する戦略 MiniMax 法で使う「とても重要な」関数です）

ソースコード 1.10　局面を評価する関数：Tic Tac Toe

```
    int evaluate(int turn){
        // 局面を評価する（MINIMAX法が使う）
        int i;
        // Checking for Rows
        for( i=0; i<7; i+=3 ){
            if( line(i,i+1,i+2) ){
                if( stones[i].getColor()==WHITE ) return +10;
                else if( stones[i].getColor()==BLACK ) return -10;
                else return 0;
            }
        }
        // Checking for Columns
        for( i=0; i<3; i++ ){
            if( line(i,i+3,i+6) ){
                if( stones[i].getColor()==WHITE ) return +10;
                else if( stones[i].getColor()==BLACK ) return -10;
                else return 0;
            }
        }
        // Checking for Diagonals
        if( line(0,4,8) ){
            if( stones[0].getColor()==WHITE ) return +10;
            else if( stones[0].getColor()==BLACK ) return -10;
            else return 0;
        }
        if( line(2,4,6) ){
            if( stones[2].getColor()==WHITE ) return +10;
            else if( stones[2].getColor()==BLACK ) return -10;
            else return 0;
        }
        return 0;
    }
```

1.6　プレイヤーのクラス

プレイヤークラスが持っているのは、プレイヤーの氏名、そのプレイヤーが採ろうとしている戦略、そのプレイヤーが置こうとしている石の種類（MARU か BATSU か）です

putStone(Board) の処理では、
プレーヤーが選んでいる戦略（ Senryaku ）を使って決めた手 te=Te()（＝石の置き場所）を、
引数で受け取ったその時の盤面（ board ）に置けるのかどうかを確認（ canPut(Stone) ）して、
置けるならば、盤面にその石を置きます（ board.setBoard(Stone) ）

この Player クラスは、戦略クラス（class Strategy）を継承しています

ソースコード 1.11　プレイヤークラス：Tic Tac Toe

```
1  package tictactoe;
2  import static tictactoe.Constants.*;
3
4  public class Player extends Strategy {
5
6      private String Name = null;
7      private int Senryaku = 0;
8      private int Color = EMPTY;
9
10     Player(int i, String n, int s) {
11         Color = i;
12         Name = n;
13         Senryaku = s;
14     }
15
16     void putStone(Board board) {
17         Stone s=null;
18         do {
19             int te = Te(board);
20             s = new Stone(te, Color);
21             if ( board.canPut(s) ) {
22                 break;
23             }
24         } while (true);
25         board.setBoard( s );
26     }
27
28     private int Te(Board board) {
29         int v = 0;
30         switch (Senryaku) {
31             case PROMPT:  v = prompt(Name); break;
32             case RANDOM:  v = random(Name); break;
33             case MINIMAX: v = bestMoveMM(board, this); break;
34             case ALPHABETA:   break;
35             case MONTECARLO:  break;
36         }
37         return v;
38     }
39
40     int getColor(){
41         return Color;
42     }
```

```
43 }
```

1.7　戦略クラス（MiniMax）

Player クラスの親クラスです

ソースコード 1.12　戦略クラス：Tic Tac Toe

```
1  package tictactoe;
2
3  import static tictactoe.Constants.*;
4  import java.io.BufferedReader;
5  import java.io.InputStreamReader;
6
7  public class Strategy {
8      int prompt(String name) {
9          BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
10         int num = -1;
11         String fmt = null, instr = null;
12         do {
13             fmt = String.format("\n石を置く場所を番号で指定（ %s の番です ）：",
                   name);
14             System.out.print(fmt);
15             try {
16                 instr = br.readLine();
17                 num = Integer.parseInt(instr);
18             } catch (Exception e) {
19                 e.printStackTrace();
20             }
21             if (!(0 <= num && num < NXN * NXN)) {
22                 System.out.println("0〜" + (NXN * NXN - 1) + " の番号を指定して");
23                 continue;
24             }
25             break;
26         } while (true);
27         return num;
28     }
29
30     int random(String name) {
31         System.out.printf("。。。 %s 思考中 。。。", name);
32         return (int) (Math.random() * (NXN * NXN));
33     }
34
35     int max(int a, int b){
36         if(a>b)return a;
37         return b;
38     }
39     int min(int a, int b){
40         if(a>b)return b;
41         return a;
42     }
43
44     private int minimax(Board board, int depth, int turn) {
45         // minimax
46     }
47
```

```
48      int bestMoveMM(Board board, Player player) {
49          // minimax
50      }
51  }
```

（以下は、上記の//minimax、の部分に納める関数です）

ソースコード 1.13　戦略クラス-minimax 法：Tic Tac Toe

```
1       private int minimax(Board board, int depth, int turn) {
2           int best, MaximizingPlayer=WHITE;
3           if (board.isWin() || board.isDraw() || depth <= 0) {
4               int score = board.evaluate(turn);
5               return score;
6           }
7           if (turn == MaximizingPlayer){
8               best = Integer.MIN_VALUE;
9               for( int i=0; i<NXN*NXN; i++){
10                  Stone s = new Stone(i, turn);
11                  if(board.canPut(s)){
12                      board.setBoard(s);
13                      int value = minimax(board, depth-1, -turn);
14                      board.setEmpty(i);
15                      best = max(best, value);
16                  }
17              }
18              return best;
19          }else{
20              best = Integer.MAX_VALUE;
21              for( int i=0; i<NXN*NXN; i++){
22                  Stone s = new Stone(i, turn);
23                  if(board.canPut(s)){
24                      board.setBoard(s);
25                      int value = minimax(board, depth-1, -turn);
26                      board.setEmpty(i);
27                      best = min(best, value);
28                  }
29              }
30              return best;
31          }
32      }
33
34      int bestMoveMM(Board board, Player machine) {
35          int bestEval = Integer.MIN_VALUE;
36          int bestMove = -1;
37          for (int i = 0; i < NXN * NXN; i++) {
38              Stone s = new Stone(i, machine.getColor());
39              if (board.canPut(s)) {
40                  board.setBoard(s);  //BLACKに打たせるには -player.getColor()の負号をとり
41                  int eval = minimax(board, 8, -machine.getColor());
42                  board.setEmpty(i);  //Board.evaluate で -1 と 1 を入れ替える
43                  if (eval > bestEval) {
44                      bestEval = eval;
45                      bestMove = i;
46                  }
47              }
48          }
```

```
49          return bestMove;
50      }
```

この MiniMax 法は、最後（勝敗がつく）まで読み切るので、決して「負けることはありません」

人間が、最善の手を打った場合には「引き分け」になります

盤面が 3 ☒ 3 と、小さいので MiniMax で最後まで読み切ることができるわけですが、盤面が大きくなると、「アルファ・ベータ・刈り」が必要になります。

1.8　最終的な完成形

最終的なソースは、tictactoe フォルダの中に納めています。

```
% ls -l tictactoe
total 56
-rw-r–r– 1 mat staff 3780 2 20 14:56 Board.java
-rw-r–r– 1 mat staff 691 2 20 14:01 Constants.java
-rw-r–r– 1 mat staff 1461 2 20 14:44 Game.java
-rw-r–r– 1 mat staff 970 2 20 14:11 Player.java
-rw-r–r– 1 mat staff 363 2 20 14:24 Stone.java
-rw-r–r– 1 mat staff 2657 2 20 14:59 Strategy.java
-rw-r–r– 1 mat staff 350 2 20 14:40 TicTacToe.java
```

参考文献

[1]