プログラム言語 B
勝敗判定付きオセロ

2023 年 12 月 27 日 (月)

指導教員：武田先生

6321120

横溝　尚也

1　通常課題

1.1　課題内容

＜拡張の手順＞
（1）オセロの盤面は緑色で、8*8 の正方形のマス目とする。
（2）○×を白黒に変更する。
（3）盤面中央の 4 マスに黒石と白石を 2 つずつ置く。
（4）初手は黒が打ち、黒と白が交互に空きマスに自分の色の石を打つ。
なお、石を置きうる場所の判定は行わなくて良い（人に任せる）。
（5）挟んだ相手の色の石を裏返して自分の色に変える。
（6）挟める石がなければパスとなり、相手の手番になる。
なお、パスの判定は自動でも手動でもどちらでも良い。
（7）盤上の全てのマスが石で埋まって空きマスがなくなれば、ゲーム終了
（終局）となる。空きマスがあっても、両者ともに挟める石がないとき
も終局となる。
（8）ゲームが終了したら黒石・白石の数を数え、多いほうが勝ちとなる。
同数の場合は引き分け。
レポートにはプログラムのソースコードを貼り付け、さらにプログラムの説明
をして下さい。拡張子 java のプログラムソースコードファイルは提出する必要
はありません。実行している画面のスクリーンショット２枚を貼り付けてくだ
さい。例題 5-3-a,b,c を基にしてそれを拡張したプログラムとすること。
レポートは１つの PDF ファイルとして LETUS にアップロードして下さい。

1.2　プログラムの内容

以下は拡張 (1)〜(8), チャレンジ (1) を実装したコードである。説明の便宜上、左に行数を振ってある。

プログラム 1　Osero.java

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4  import java.util.*;
5
6  public class Osero extends JPanel {
7      static final int EMPTY = 0, Kuro = 1, Shiro = 2;
8      //拡張 (1)マスにする 8*8
9      static final int YMAX = 8, XMAX = 8;
10     ArrayList<Figure> figs = new ArrayList<Figure>();//図形オブジェクトの保持
11     boolean turn = true;//黒が true
12     int winner = EMPTY;
```

```
13      int[][] board = new int[YMAX][XMAX];
14      Text t1 = new Text(20, 20, "オセロ、次の手番：黒
            ", new Font("Serif", Font.BOLD, 22));
15
16      public Osero() {
17          figs.add(t1);
18          for (int i = 0; i < YMAX * XMAX; ++i) {
19              int r = i / YMAX, c = i % XMAX;
20              //拡張(1)盤面を緑にする
21              figs.add(new Rect(Color.GREEN, 80 + r * 30, 40 + c * 30, 28, 28));
22          }
23
24          //拡張(3)初期石の配置
25          for (int i = 0; i < 4; i++) {
26              //初期石の置きたいマスに対して配列インデックスを指定
27              int index = i == 0 ? 28 : i == 1 ? 29 : i == 2 ? 36 : 37;
28              Figure targetFigure = figs.get(index);
29              Rect targetRect = (Rect) targetFigure;
30              int targetX = targetRect.getX();
31              int targetY = targetRect.getY();
32
33              // i の値に応じてaddKuroishi またはaddShiroisi を実行
34              if (i == 0 || i == 3) {
35                  figs.add(new Kuroishi(targetX, targetY,10));
36                  if(i == 0){
37                      board[3][3] = Kuro;
38                  }else{
39                      board[4][4] = Kuro;
40                  }
41              } else {
42                  figs.add(new Shiroisi(targetX, targetY,10));
43                  if(i == 1){
44                      board[3][4] = Shiro;
45                  }else{
46                      board[4][3] = Shiro;
47                  }
48              }
49          }
50          setOpaque(false);
51
52          //大きく変更。拡張(4~8)
53          addMouseListener(new MouseAdapter() {
54              public void mouseClicked(MouseEvent evt) {
55                  Rect r = pick(evt.getX(), evt.getY());
56                  int x = (r.getX() - 80) / 30, y = (r.getY() - 40) / 30;
57                  if (isValidMove(x, y)) {
```

```
58                      // オセロを置く処理
59                      board[y][x] = (turn ? Kuro : Shiro);
60                      if(turn){
61                          figs.add(new Kuroishi(80 + 30 * x, 40 + 30 * y,10));
62
63                      }else{
64                          figs.add(new Shiroisi(80 + 30 * x, 40 + 30 * y,10));
65                      }
66                      flipOpponentStones(x, y);
67                      turn = !turn;
68                      t1.setText((turn ? "黒" : "白") + "のターン");
69                      repaint();
70                  } else if(board[y][x] != EMPTY){
71                      t1.setText((turn ? "黒" : "白") + "のターンそこは空いていません:");
72                      repaint();
73                      return;
74                  }else{
75                      t1.setText((turn ? "黒" : "白") + "のターンそこには石を置けません:");
76                      repaint();
77                      return;
78                  }
79
80                  finOrSkip();
81                  repaint();
82
83              }
84          });
85      }
86
87      //拡張(4)
88      //石が置けるか判定
89      private boolean isValidMove(int x, int y) {
90          if (board[y][x] != EMPTY) {
91              return false;
92          }
93          int opponentStone = turn ? Shiro : Kuro;
94
95          // 方向それぞれに対してチェック8
96          for (int dx = -1; dx <= 1; dx++) {
97              for (int dy = -1; dy <= 1; dy++) {
98                  if (dx == 0 && dy == 0) continue;//スキップ
99                  if (canFlip(x, y, dx, dy, opponentStone)) {
100                     return true;
101                 }
102             }
103         }
```

```
104         return false;
105     }
106
107     //拡張（4）
108     //ある方向に対して反転できるか判定
109     private boolean canFlip(int x, int y, int dx, int dy, int opponentStone) {
110         int nx = x + dx;
111         int ny = y + dy;
112         boolean hasOpponentStone = false;
113
114         while (nx >= 0 && nx < XMAX && ny >= 0 && ny < YMAX) {
115             if(board[ny][nx] == (turn ? Kuro : Shiro)) {
116                 if(hasOpponentStone){
117                     return true;
118                 }
119                 break;
120             }else if (board[ny][nx] == EMPTY) {
121                 break;
122             }else{
123                 hasOpponentStone = true;
124                 nx += dx;
125                 ny += dy;
126             }
127         }
128         return false;
129     }
130
131     //拡張（5）
132     //石の反転
133     private void flipOpponentStones(int x, int y) {
134         int myStone = (turn ? Kuro : Shiro);
135         int opponentStone = (turn ? Shiro : Kuro);
136
137         // 上下左右および斜めの方向に相手の石が続いているかどうかを判定し、反転させる
138         flipDirection(x, y, 1, 1, myStone, opponentStone);
139         flipDirection(x, y, 1, -1, myStone, opponentStone);
140         flipDirection(x, y, 1, 0, myStone, opponentStone);
141         flipDirection(x, y, 0, 1, myStone, opponentStone);
142         flipDirection(x, y, -1, 1, myStone, opponentStone);
143         flipDirection(x, y, -1, -1, myStone, opponentStone);
144         flipDirection(x, y, -1, 0, myStone, opponentStone);
145         flipDirection(x, y, 0, -1, myStone, opponentStone);
146     }
147
148     //拡張（5）
149     //ある方向に対しての石の反転
```

```
150     private void flipDirection(int x, int y, int dx, int dy, int myStone, int
            opponentStone) {
151         int nx = x + dx;
152         int ny = y + dy;
153         ArrayList<Point> stonesToFlip = new ArrayList<>();
154 
155         while (nx >= 0 && nx < XMAX && ny >= 0 && ny < YMAX && board[ny][nx] ==
                opponentStone) {
156             stonesToFlip.add(new Point(nx, ny));
157             nx += dx;
158             ny += dy;
159         }
160 
161         if (nx >= 0 && nx < XMAX && ny >= 0 && ny < YMAX && board[ny][nx] == myStone)
                 {
162             for (Point p : stonesToFlip) {
163                 board[p.y][p.x] = myStone;
164                 if(myStone == Kuro){
165                     figs.add(new Kuroishi(80 + 30 * p.x, 40 + 30 * p.y,10));
166                 }else{
167                     figs.add(new Shiroisi(80 + 30 * p.x, 40 + 30 * p.y,10));
168                 }
169             }
170         }
171     }
172 
173     //拡張(7,8)
174     public void finOrSkip(){
175         //石の数が個になる、または全部石で埋まったら対局終了0
176         if(!exIshi(turn ? Kuro : Shiro) || !exEmpty()){
177             t1.setText("試合終了勝ったのは!..." + decideWin());
178             return;
179         }
180 
181         //拡張(6)チャレンジ,(1)
182         //で置けるマスがあるか判定し、ないなら自動的に手番スキップIsPut
183         if(!isPut()){
184             turn = !turn;
185             t1.setText("置けるマスがないのでスキップ。次の手番：" + (turn ? "黒" : "白
                    "));
186         }
187     }
188 
189 
190     //拡張(6)チャレンジ,(1)
191     //置けるマスが一つでも存在するか判定
```

```
192     public boolean isPut(){
193         for(int i = 0; i < YMAX * XMAX; ++i){
194             int r = i / YMAX, c = i % XMAX;
195             if(board[r][c] == EMPTY && isValidMove(c,r)){
196                 return true;
197             }
198         }
199         return false;
200     }
201
202     //拡張 (7)
203     //盤面に自分の石が存在するか
204     public boolean exIshi(int mycolor){
205         for(int i = 0; i < YMAX * XMAX; ++i){
206             int r = i / YMAX, c = i % XMAX;
207             if(board[r][c] == mycolor){
208                 return true;
209             }
210         }
211         return false;
212     }
213
214     //拡張 (7)
215     //盤面に空きマスが存在するか
216     public boolean exEmpty(){
217         for(int i = 0; i < YMAX * XMAX; ++i){
218             int r = i / YMAX, c = i % XMAX;
219             if(board[r][c] == EMPTY){
220                 return true;
221             }
222         }
223         return false;
224     }
225
226     //拡張 (8)
227     //勝者の判別
228     public String decideWin(){
229         int countShiro = 0, countKuro = 0;
230         for(int i = 0; i < YMAX * XMAX; ++i){
231             int r = i / YMAX, c = i % XMAX;
232             if(board[r][c] == Shiro){
233                 countShiro += 1;
234             }else if (board[r][c] == Kuro){
235                 countKuro += 1;
236             }
237         }
```

```
238         if(countShiro > countKuro){
239             return "白!";
240         }else if(countShiro < countKuro){
241             return "黒!";
242         }else{
243             return "引き分け!";
244         }
245     }
246 
247     public Rect pick(int x, int y) {
248         Rect r = null;
249         for (Figure f : figs) {
250             //クラスのインスタンスであるか判定 Rect
251             //与えられた座標が四角形の中にあるかどうかの判定
252             if (f instanceof Rect && ((Rect) f).hit(x, y)) {
253                 r = (Rect) f;//にオブジェクトを代入 rRect
254             }
255         }
256         return r;
257     }
258 
259     public void paintComponent(Graphics g) {
260         for (Figure f : figs) { //リスト内のオブジェクトの描画 figs
261             f.draw(g);
262         }
263     }
264 
265     private int ck(int x, int y, int dx, int dy) {
266         int s = board[y][x], count = 1;//取得した状態 (EMP,BATU,MARU)の連続数をカウント
267         for (int i = 1; ck1(x + dx * i, y + dy * i, s); ++i) {
268             ++count;
269         }
270         for (int i = 1; ck1(x - dx * i, y - dy * i, s); ++i) {
271             ++count;//逆方向
272         }
273         return count;
274     }
275 
276     private boolean ck1(int x, int y, int s) {
277         return 0 <= x && x < XMAX && 0 <= y && y < YMAX && board[y][x] == s;
278     }
279 
280     public static void main(String[] args) {
281         JFrame app = new JFrame();
282         app.add(new Osero());
283         app.setSize(500, 320);
```

```
284         app.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
285         app.setVisible(true);
286     }
287 
288     interface Figure {
289         public void draw(Graphics g);
290     }
291 
292     static class Text implements Figure {
293         int xpos, ypos;
294         String txt;
295         Font fn;
296 
297         public Text(int x, int y, String t, Font f) {
298             xpos = x;
299             ypos = y;
300             txt = t;
301             fn = f;
302         }
303 
304         public void setText(String t) {
305             txt = t;
306         }
307 
308         public void draw(Graphics g) {
309             g.setColor(Color.BLACK);
310             g.setFont(fn);
311             g.drawString(txt, xpos, ypos);
312         }
313     }
314     //拡張 (2)○×を白黒にする
315     static class Kuroishi implements Figure {
316         int xpos, ypos, size;
317 
318         public Kuroishi(int x, int y, int s) {
319             xpos = x;
320             ypos = y;
321             size = s;
322         }
323 
324         public void draw(Graphics g) {
325             g.setColor(Color.BLACK);
326             ///((Graphics2D) g).setStroke(new BasicStroke(4));
327             g.fillOval(xpos - size, ypos - size, 2 * size, 2 * size);
328         }
329     }
```

```
330     //拡張(2)○×を白黒にする
331     static class Shiroisi implements Figure {
332         int xpos, ypos, size;
333
334         public Shiroisi(int x, int y, int s) {
335             xpos = x;
336             ypos = y;
337             size = s;
338         }
339
340         public void draw(Graphics g) {
341             g.setColor(Color.WHITE);
342             //((Graphics2D) g).setStroke(new BasicStroke(4));
343             g.fillOval(xpos - size, ypos -size, 2* size, 2 * size);
344         }
345     }
346
347     static class Rect implements Figure {
348         Color col;
349         int xpos, ypos, width, height;
350
351         public Rect(Color c, int x, int y, int w, int h) {
352             col = c;
353             xpos = x;
354             ypos = y;
355             width = w;
356             height = h;
357         }
358
359         public boolean hit(int x, int y) {
360             return xpos - width / 2 <= x && x <= xpos + width / 2 && ypos - height /
                    2 <= y && y <= ypos + height / 2;
361         }
362
363         public int getX() {
364             return xpos;
365         }
366
367         public int getY() {
368             return ypos;
369         }
370
371         public void draw(Graphics g) {
372             g.setColor(col);
373             g.fillRect(xpos - width / 2, ypos - height / 2, width, height);
374         }
```

```
375     }
376 }
```

1.3 プログラムの説明

拡張手順に沿ってプログラムを説明していく。ただし、既存のソースコードの箇所は説明を省略する。

1.3.1 拡張 1.

盤面の色を緑にし、マスを 8×8 に変更した。9 行目の YMAX,XMAX をともに 8 とし、21 行目の長方形の描画の際の Color を GREEN とした。

1.3.2 拡張 2.

○×を白黒に変更した。14,68,71,75 行目のテキストの描画を白黒にし、324,340 行目の maru,batsu クラスを Shiroisi,Kuroishi クラスに変名して draw メソッドを白黒で円を塗りつぶす内容に変更した。塗りつぶしは、fillOval を用いた。

1.3.3 拡張 3.

盤面中央の４マスに初期石を置くプログラムを 24〜49 行目に記述した。４つの石を for 文を使用して石の描画を行った。この時、先ほどの Kuroishi,Shiroisi メソッドを使って描画した。またこの時、board[][] 配列の状態を適宜変更し、盤面の状態を配列に記述している。細かいプログラムについては、基本的に元のプログラムと同じである。

1.3.4 拡張 4.

次にゲームの基本的な機能の付与を行った。53〜129 行目がプログラムの主な変更箇所である。

- ５６行目　8×8 への盤面の変更に伴って座標を調節
- ５７〜７８行目　isValidMove() でクリックされたマスに石を置くことができるか判定し、58〜69 行目が石を置くときの処理、70〜73 行目がすでにマス目の状態が!=EMPTY であり置くことができないとき、74〜78 行目が挟める石がなく置けないときの処理である。66 行目にある flipOpponentStone() は、拡張 (5) で実装する石の反転を行うメソッドである。
- ８９〜１０３行目　57 行目で引数として受け取ったマスの座標に石が置けるか否かを判定するプログラムである。90 行目でマスが EMPTY でないときはおけないため false を返し、93 行目で相手の色を opponentStone に格納 (true が黒、false が白で初めに宣言している。)。96 行目以降で判定するマスに対して全８方向に対して１方向でもひっくり返せる石があれば true, 一つもなければ false を返す記述である。また、99 行目でのある１方向に対してひっくり返す石が存在するかどうかの判定は canFlip() に別で記述して簡潔化している。
- １０９〜１２９行目　99 行目で使用するメソッド canFlip メソッドについての記述である。引数で対象のマスの座標 x,y と、どの方向に対して判定するのかを表す dx,dy を受け取る。112 行目で hasOpponentStone を false として初期宣言している。これは、１方向に対して EMPTY がなく、自分の色が存在すれば挟めるが１つ以上は相手の色が存在してないといけないため、相手の石が確認できた

ら、hasOpponentStone を true に変更し、hasOpponentStone が true かつ自分に石が観測すれば挟めると判定する。

1.3.5 拡張 5.

挟んだ相手の色をひっくり返す機能の実装を行った。66 行目で flipOpponentStone() として実行している。

- １３３～１４６行目　flipOpponentStone() の記述である。134,135 行目で自分の色と相手の色をそれぞれ myStone,opponentStone に格納し、138～145 では 1 方向に対して石の反転を行う flipDirection() を全 8 方向に対して実行している。
- １５０～１７１行目　flipDrection() の記述である。引数でマスの座標 x,y と方向に対する情報 dx,dy を受け取る。盤面が続き、かつ自分の色があらわれるまでの間の盤面情報を相手の色から自分の色へと変更し、自分の色の石を再描画して反転させている。

1.3.6 拡張 6.

パスの判定の自動化を行った。チャレンジ課題 (1) も並行して作成したため、ここで説明を行う。80 行目でクリックしたマスに対する処理を行った後に、finOrSkip() を用いて拡張 (7) で行う終了判定とパス判定をこのメソッドで行っている。

- １７４～１８７行目　前述した通りパス判定と終了判定をここでまとめそれぞれの処理をしている。詳しくは、176 行目で試合終了判定を exIshi() で行い勝者をテキストで出力、183 行目でパス判定を isPut() で行いパスならばスキップして手番が変わったことをテキストで出力している。
- １９２～２００行目　スキップ判定メソッド isPut() についての記述である。あるマスが EMPTY でありかつ isValidMove() を用いて置けるマスであるかどうかを判定してそれを全マスにおいてループしている。

1.3.7 拡張 7.

ゲームの終了判定プログラムを実装した。拡張 (6) と同様に 80 行目で使用する finOrSkip() 内 (174～187 行目) の 176 行目で終了判定プログラムを実装している。自分の石が一つもないまたは、マスがすべて埋まっているのどちらかに当てはまればゲーム終了となる。

- ２０４～２１２行目　自分の色が盤面に存在するか判定する exIshi() についての記述である。全マスについて mycolor が存在すれば true, 一つも存在しないなら false を返す。
- ２１６～２２４行目　盤面に空きマスが存在するか判定する exEmpty() についての記述である。全マスについて EMPTY 状態のマスがあれば true, なければ false を返す。

1.3.8 拡張 8.

ゲーム終了判定を拡張 (7) で実装し、この時の勝者を計算するプログラムを実装した。174 行目の finOrSkip() 内で試合終了判定が下されたときに decideWin() で勝者を計算して、白か黒を返す。

- ２２８～２４５行目 decideWin() についての記述である。全マスについて状態が白であれば

countShiro をインクリメントし、黒であれば countKuro をインクリメントする動作をループする。最後に countShiro と countKuro を比較して勝者または引き分けを返す。

1.4 実行結果

以下が対局中の実行過程である。課題内容には２枚のスクリーンショットとありましたがすべての状態を示すために指示よりも多くの画像を添付しました。ご容赦ください。

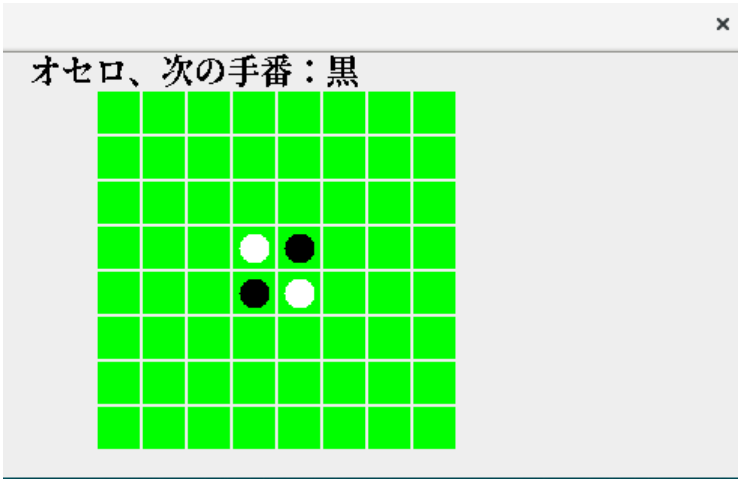

図１　対局開始時

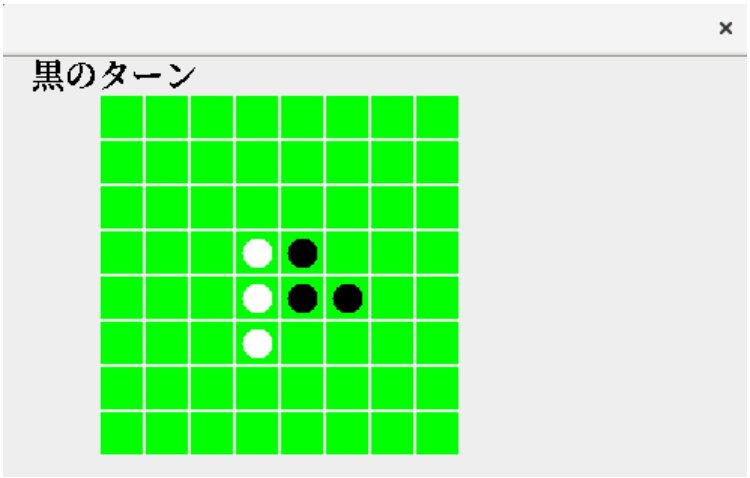

図２　正常にゲームが進行しているとき

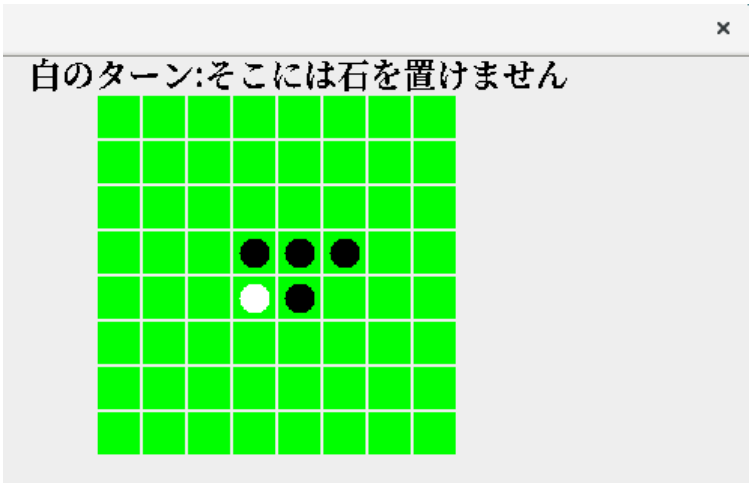

図３　おけない場所を指定したとき

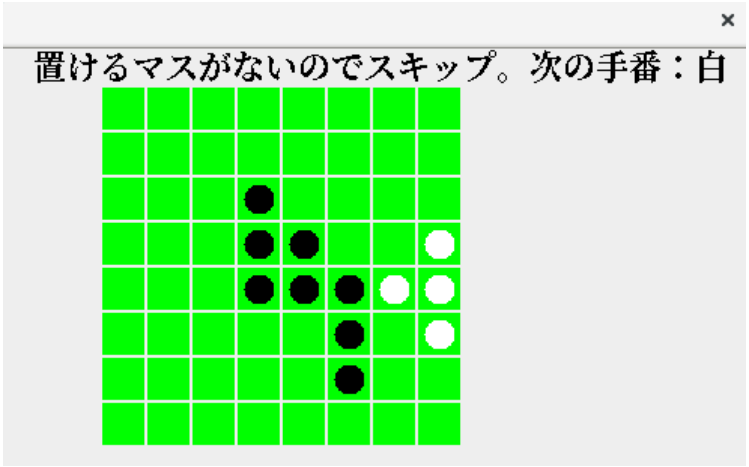

図４　スキップの状態のとき

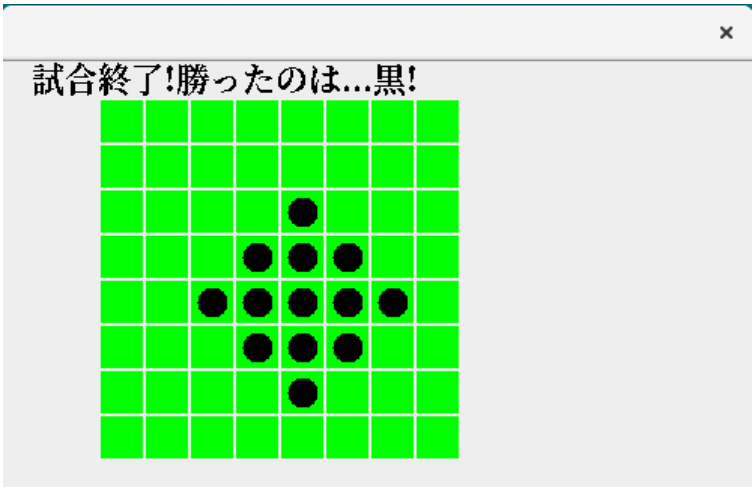

図５　自分の石がなくなったとき

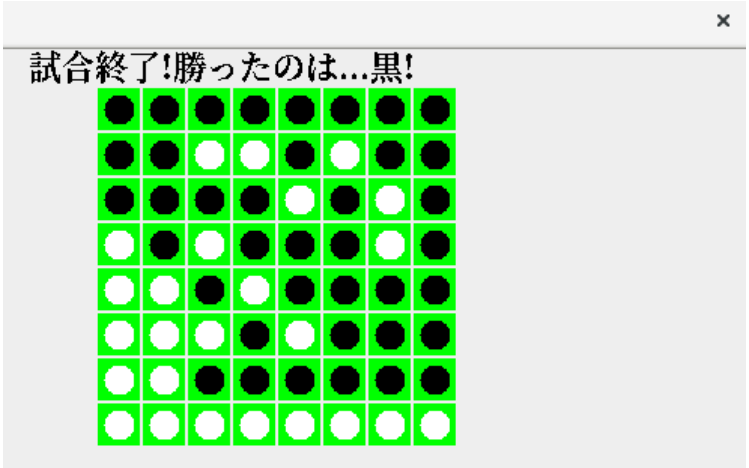

図 6　すべてのマスが埋まったときの対局終了時

2　チャレンジ課題

2.1　課題内容

＜チャレンジ課題（必須ではありません＞
（１）パスの判定の自動化
（２）AI の作成
(2-1) 石を置きうる場所の候補を求める。
(2-2) 候補の中から 1 個を選択（簡単にはランダムに選択）する。

パス判定の自動化については、通常課題の方のプログラムで既に実装済みである。また、石を置きうる場所の候補の算出は、通常課題の終局判定や、パス判定ですでに実装されており、このチャレンジ課題の拡張機能でもこの機能を使用した。ここでは白の手番のみ AI による対局、黒は今まで通りマウスイベントによる動作とし、AI を作るうえでの拡張した個所を中心に説明する。

2.2　プログラムの内容

基本的には上記に書いたプログラムをもとに拡張した。変更箇所を指揮に記す。
82 行目・・・以下のプログラムを追加し、白の手番では putAI() を実行し、自動で手を決め、描画するプログラムを追記した。

プログラム 2　OseroAI.java

```
1 //白のターンは AI で自動化
2 putAI();
3 finOrSkip();
4 return;
```

また、putAI() の記述は以下のようにした。今回採用したアルゴリズムは、置けるマスを全て算出し、配列に

そのマスのインデックスを格納し、ランダムメソッドでランダムにマスに置くようにした。

プログラム 3　OseroAI.java

```
1  //AI による実装
2  public void putAI(){
3      try {
4          Thread.sleep(3000); // 人が過程を追いやすいように３秒スリープ
5      } catch (InterruptedException e) {
6          // が発生した場合の処理 InterruptedException
7          e.printStackTrace();
8      }
9      if(turn){
10         t1.setText("エラーが発生しました");
11     }else{
12         //配置可能なマスを全て抽出
13         int currentArray = 0; //の indexArraylength
14         int[][] indexArray = new int[64][2]; //配置可能なマスのインデックスを格納する
               配列
15         for(int i = 0; i < YMAX * XMAX; ++i){
16             int r = i / YMAX, c = i % XMAX;
17             if(board[r][c] == EMPTY && isValidMove(c,r)){
18                 indexArray[currentArray][0] = r;
19                 indexArray[currentArray][1] = c;
20                 currentArray += 1;
21             }
22         }
23
24         //ランダムで置く場所を選択
25         int randomNumber = (int) (Math.random() * (currentArray + 1));
26         int y = indexArray[randomNumber][0];
27         int x = indexArray[randomNumber][1];
28         board[y][x] = Shiro;
29         figs.add(new Shiroisi(80 + 30 * x, 40 + 30 * y,10));
30         flipOpponentStones(x, y);
31         turn = !turn;
32         t1.setText((turn ? "黒" : "白") + "のターン");
33         repaint();
34     }
35 }
```

2.3　プログラムの説明

- ３～８行目　AI 側 (白) の手番はプログラムにより一瞬で終了してしまうため、ゲームの流れがわかりやすいように sleep() を使用してスリープした。
- １２～２２行目　１３行目で石を置けるマスの数をカウントするための currentArray を、１４行目で

置けるマスのインデックスをか保存するための配列 indexArray を宣言している。１５行目で全マスについて置けるマスであるか isValidMove() で判定し currentArray,indexArray を更新している。

- ２５～３３行目　currentArray 個の置けるマスのうちランダムに選び、選ばれたマス board[y][x] の状態を自分の色に更新し、flipOpponentStones() で石が置かれたときの処理を実行している。

3　考察

3.1　AI のアルゴリズムについて

今回はオセロゲームを実装することを主に行い、AI 作成については一番簡単なアルゴリズムであるランダムに置く場所を選択した。しかし、これではオセロゲームにおいて強い AI であるとは言えない。ここで実装できなかったより強いオセロ AI のアルゴリズムをここで考察する。

3.1.1　重みづけを事前に行う [1]

オセロには置いたら強いマスや置いたらいけないマスが存在するボードゲームである。この特性を生かして、盤面 64 マスに事前に重みづけをした配列を用意し、それに基づいて置くことのできるマスのうち、最もポイントの高いマスに置くことを採用するアルゴリズムである。(例えば四つ角は最も重みづけを大きくする。)

しかし、これは盤面の状態を考慮しない、初めから決められた値である。よっていくつもの観点から重さ付けの値を用意し、総合的な値をその場その場で計算することで最適解を導出できるのではないかと考えた。

例えば、自分が今置いたことによるひっくり返せたマスが次の一手で返されたら、今の一手は無意味ということになる。そこで次の一手で返される場所には値を低くしていくなどの改善法もあると考えた。

3.1.2　Minimax（ミニマックス）探索法について

自分が最も有利となるような手を打つことだけを考えるのではなく、相手が自分にとって最も不利になるよう打ってくることを想定し、その中で自分がどの手を打てばいいか最良の選択肢を採択するという方法である。置きたい候補のマスすべてに対して 2,3 手先の盤面すべてを試しつつ、相手の打つ石に対しても評価を行い、最終的な３手先の状況が最善となるものを選ぶ。

この方法では試す通りが莫大な量であるため、１手を導くまでの計算時間がとてつもなく長いことが想像できる。この Minimax 探索法の計算量を減らしたものがαβ法である。

3.1.3　αβ法について [2]

基本的にはミニマックス探索法と同じアルゴリズムである。違う点は計算量を減らすためのアルゴリズムである。相手は自分が最も不利になるような手を打つことを想定しているため、相手の打つ手の中で自分にとっての評価値が最小なもの以外の手を考えない（計算しない）ことにする。これにより、計算量が大幅に減少する。

4　参考文献

参考文献

[1]　わかばテクノロジ
http://wakaba-technica.sakura.ne.jp/lab/lab_othello.html
[2] オセロゲーム開発 〜アルファベータ法（alpha-beta search）
https://uguisu.skr.jp/othello/alpha-beta.html