情報科学演習_第３期課題
ネットワークプログラミング

6321120
横溝　尚也

提出日：7 月 11 日（火）

第 1 章

課題 1

1.1 課題内容

プロトコルに準拠した Othello の AI クライアントを作成せよ。
AI クライアントは起動以外に操作することなくゲーム終了まで自動で行えるプログラムとする．
以下が Otheloo 通信プロトコルである。

NICK メッセージ

クライアントからサーバに送られるメッセージでチャット等に使われるユーザのニックネームを変更する．$NICKusername$ のように送るとユーザのニックネームが username に切り替わる．このメッセージはゲームに中にいつでも何度もでも受け付ける．クライアントから NICK メッセージが来るまでは，サーバはそれぞれ BLACK,WHITE というニックネームを使用する．

PUT メッセージ

クライアントが石を置く座標を送信する．$PUT32$ この例では，横方向に 4 番目，下方向に 3 番目の盤上へ，自身の色の石を置く命令になる． PUT 命令が正常に処理されると，後述する BOARD メッセージと TURN メッセージが両方のクライアントに送られる．正常に処理できない場合は ERROR の後に理由の番号がつけられ，クライアントに送られる．
ERROR 1
書式が間違っているエラー
ERROR 2
PUT 命令で指定した盤目に自身の色の石が置けないエラー
ERROR 3
相手のターンの時に PUT 命令が送られた場合のエラー
ERROR 4
処理できない命令が送られた場合のエラー

SAY メッセージ

チャットメッセージを送る．$SAYhello$ と送信すると，サーバから $SAY < username >: hello$ という文

字列が，双方のクライアントに送信される．

NICK メッセージチャットメッセージを送る．$SAYhello$ と送信すると，サーバから SAY < $username$ >: $hello$ という文字列が，双方のクライアントに送信される．

BOARD メッセージ

サーバからクライアントへは BOARD 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 -1 0 0 0 0 0 0 -1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 のようにスペース区切りで 8x8 の盤情報が送られる．1 は黒石，-1 は白石，0 はまだ石が置かれていない．先頭から順に左列 8 マス，2 列目 8 マス・・・8 列目 8 マスが左から順に送られる．クライアントから BOARD メッセージが送られると，サーバは上記の情報を返すが， 盤に新しい石が置かれた場合などはサーバから双方のクライアントへ BOARD メッセージが送られる．

TURN メッセージ

サーバからクライアントへ送られるメッセージで現在の置く順番になっている色を指定する．$TURN1$ の場合は，黒石を置く順番である． 基本的には PUT するごとに入れ替わるが，置けるところが無い場合は，連続で同じ色のターンになることもある．

END メッセージ

サーバがゲーム終了を告げるメッセージ．END の後に勝敗や石の数が告げられる．

START メッセージ

ゲーム開始時にサーバは START メッセージを双方のクライアントに送る．その際に 1 か-1 の文字列をクライアントに送信する．1 の場合は黒，-1 の場合は白を担当するクライアントであることを告げている．以下の例は，黒石を担当するクライアントに送られるメッセージである．$START1$

CLOSE メッセージ

相手の切断を知らせるメッセージで，何らかのネットワークエラーでクライアントが切断された場合，サーバから相手のクライアントに送られる．また，クライアントからサーバへ送った場合は，送ったクライアントとサーバの TCP コネクションが切断され，サーバから相手のクライアントに CLOSE メッセージが送られる．

1.2 アルゴリズムの説明

1.2.1 Othello 通信プロトコル

サーバーとのメッセージのやり取りは課題内容に記述されている通りプログラムを行う。

1.2.2 受け取った盤面から石の置ける場所の検索

サーバーから送られてきた盤面の状況から初めに自分が置ける場所を探す必要がある。あるマスにおいておくことができるのか否か判断するには、そのマスの周囲８方向のうち１方向でも石をひっくり返せればよい。よって各方向に対してひっくり返せる石があるのか調べていく。例えば自分が黒、相手が白の場合、調べる方向に対して置きたい場所から白、白、黒となっている場合、置くことができる。その方向に対してひっくり返すことができるか判断する条件は以下の二つである。

1. 置きたい場所の隣（調べる方向に対する）が白であること。
   （空白、白、白、黒であってもこれは置くことができない。）
2. 白から始まり連続して白が置かれ、黒が囲い込むように置かれていること。
   （白、白、白であれば黒で囲めていないため置くことはできない。）

この条件を満たす方向が８方向のうち一つでもあればそのマスには置くことはできると判断する。

1.2.3 置ける場所の中から最適解を導出

置ける場所の中からどこに置くべきか考える。おそらくこの課題において、このアルゴリズムが強いオセロ AI を作るために最も大切なアルゴリズムであると考える。
結論から言うと、理想的な強いオセロ AI となるアルゴリズムは考えることができなかった。授業資料にある MONTEAI と対戦させた結果、これより優れたアルゴリズムではなかった。

自分の採用したアルゴリズムは、あらかじめ盤面のマスごとに数値の重み付けを行い、その数値が最大であるマスに置く、というアルゴリズムである。この重み付けする数値に関しては、ネット上の数値を引用した。[*1]

[*1] 参考文献に URL を記載

1.3 プログラムの説明

説明のためにソースコードに行数をふってある。以下がソースコードである。

プログラム 1.1 ：OthelloClient.java

```
1  import java.net.*;
2  import java.io.*;
3  import java.util.*;
4
5  class OthelloClient {
6      static int c;
7      static Socket s;
8      static InputStream sIn;
9      static OutputStream sOut;
10     static BufferedReader br;
11     static PrintWriter pw;
12     static int MyColor = 0;
13     static int[] boardint = new int[64];
14     static int[][] board = new int[8][8];
15     static String[] strSer;
16     static String str;
17     static boolean[][] checkput = new boolean[8][8];
18     static int OthelloValue[][] = {{2500,767,607,642,642,607,767,2500},
19                                    {767,344,284,103,103,284,344,767},
20                                    {607,284,114,182,182,114,284,607},
21                                    {642,103,182,202,202,182,103,642},
22                                    {642,103,182,202,202,182,103,642},
23                                    {607,284,114,182,182,114,284,607},
24                                    {767,344,284,103,103,284,344,767},
25                                    {2500,767,607,642,642,607,767,2500}};
26
27     static boolean Check_Put(int g, int r, int mycolor){
28         if(board[g][r] != 0){ // 石がすでに置いてある
29             return false;
30         }
31
32         if(g < 0 || g >= 8 || r < 0 || r >= 8){ // 盤面外
33             return false;
34         }
35
36         for(int dg = -1; dg <= 1; dg++){
37             for(int dr = -1; dr <= 1; dr++){
38                 if(dg == 0 && dr == 0){ // その場所はスルー
39                     continue; // スキップ
```

```
40                }
41
42                int ng = g + dg;
43                int nr = r + dr;
44                boolean op = false;
45                while(ng >= 0 && ng < 8 && nr >= 0 && nr < 8){
46                    if(board[ng][nr] == mycolor){
47                        if(op){ // 石の反転条件
48                            return true; // の時点で終了 true
49                        }
50                        break;
51
52                    }else if(board[ng][nr] == 0){ // 空きマス
53                        break;
54                    }else{
55                        op = true;
56                        ng = ng + dg;
57                        nr = nr + dr; // 同じ方向の一つ先のマスに進む
58                    }
59                }
60            }
61        }
62        return false; // なかった場合
63    }
64
65    public static void main(String args[]) {
66
67        if (args.length != 1) {
68            System.out.println("No hostname given");
69            System.exit(1);
70        }
71
72        try {
73            s = new Socket(args[0], 46000);
74
75            sIn = s.getInputStream();
76            sOut = s.getOutputStream();
77            br = new BufferedReader(new InputStreamReader(sIn));
78            pw = new PrintWriter(new OutputStreamWriter(sOut),true);
79
80            pw.println("NICK 6321120");
81            str = br.readLine(); // 読み込み
82            strSer = str.split(" "); // 読み込んだ文字列を空白で区切り配列に格納
83
84            // ここまではチャットプログラム引用
85            while(strSer[0] != "END"){
```

```
86                  switch (strSer[0]){
87
88                  case "START":
89                      if(Integer.parseInt(strSer[1]) == 1){
90                          MyColor = 1; // 黒
91                          System.out.println(" あなたは黒です");
92
93                      }else{
94                          MyColor = -1;
95                          System.out.println(" あなたは白です");
96
97                      }
98                      break;
99
100                 case "BOARD":
101                     int i = 0, j = 0, k = 0, l = 0;
102                     while(l < 64){
103                         boardint[l] = Integer.parseInt(strSer[l+1]);
104                         l++;
105                     }
106                     System.out.println(" 現在の盤面状況");
107                     while(j < 8){
108                         i = 0;
109                         while(i < 8){
110                             board[i][j] = boardint[k];
111                             if(boardint[8*i+j] == 1){
112                                 System.out.print(" ●");
113                             }else if(boardint[8*i+j] == -1){
114                                 System.out.print(" ○");
115                             }else{
116                                 System.out.print(" ");
117                             }
118                             i++;
119                             k++;
120                         }
121                         System.out.print("\n");
122                         j++;
123                     }
124                     for(int x = 0; x < 8; x++){
125                         for(int y = 0; y < 8; y++){
126                             checkput[y][x] = Check_Put(y, x, MyColor);
127                         }
128                     }
129
130                     break;
131
```

```
132
133                 case "ERROR":
134                     switch(strSer[1]){
135                     case "1":
136                         System.out.println(" 書式が間違っています\n");
137                         break;
138                     case "2":
139                         System.out.println(" そこには置けません\n");
140                         break;
141                     case "3":
142                         System.out.println(" 相手のターン中です\n");
143                         break;
144                     case "4":
145                         System.out.println(" 処理できない命令です");
146                         break;
147                     }
148
149                 case "TURN":
150                     if(Integer.parseInt(strSer[1]) == MyColor){
151                         int x = 0;
152                         int y = 0;
153                         int value = 155;
154                         for(i = 0; i < 8; i++){
155                             for(j = 0; j < 8; j++){
156                                 if(Check_Put(j, i, MyColor)){
157                                     if(value < OthelloValue[j][i]){
158                                         y = j;
159                                         x = i;
160                                         value = OthelloValue[j][i];
161                                     }
162                                 }
163                             }
164                         }
165                         pw.println("PUT"+" "+x+" "+y);
166                         System.out.println("PUT"+" "+x+" "+y);
167                     }else{
168                     }
169                     break;
170
171
172                 case "CLOSE":
173                     System.out.println(" 相手が切断しました");
174                     break;
175                 }
176                 str = br.readLine();
177                 System.out.println(str);
```

```
178                 strSer = str.split(" ");
179             }
180             sIn.close();
181             sOut.close();
182             s.close();
183 
184         }
185 
186 
187         catch (IOException e){
188             System.err.println("Caught IOException");
189             System.exit(1);
190         }
191     }
192 }
```

1〜26 行目

インポートや、サーバとのやり取りの定義。

- 5行目　　OthelloClient を宣言。今回は一つのクラスで構成。
- １２行目　自分の色を設定する MyColor を宣言。黒を 1、白を-1 とする。
- １８行目　マスごとに重み付けした値を２次元配列に格納している。

27〜64 行目

Check_Put メソッドについての記述。引数として受け取ったマスに対して、置けるか否かを判断する。

- 28 行目　すでに石が置いてある場合、false を返す。
- 32 行目　８×８マス以外の場所が引数である場合、false を返す。
- 36 行目　上記以外の場合、全８方向に対して for 文を用いてひっくり返せる方向があるか調べる。
- 42 行目 ある方向に対して、相手の色の石が続く限りその方向に進んでいき、自分の色の石で囲めているか判断する。例えば、左上方向に対して調べるとき、$dg = -1, nr = -1$ であり、ng, nr の値が順にデクリメントされていく。

65〜184 行目

メイン関数の記述。主に Othello プロトコルに合った記述を行っている。

- 73 行目　今回自分のプログラムではポート番号 46000 と設定した。
- 84 行目まで　$UdpSend.java$ を参考にしたため説明は省略。
- 86 行目　配列 strSer に格納した初めの要素にメッセージの種類、次の要素にその内容が格納されてい

るため、switch 文を使用して strSer[0] に格納されているメッセージ内容で分岐している。

- START 自分の色をサーバーから受け取り、MyColor の値を設定する。
- BOARD　受け取った盤面状況を可視化できるよう出力している。124 行目では、Check_Put メソッドを使い、配列 checkput におけるマスを記憶している。
- ERROR エラー処理に関しては課題内容に沿ってメッセージを出力している。
- TURN　150 行目で自分の番であるときの処理を行っている。置けるマスの中で、そのマスの重み付けの値 (OthelloValue) が最大のものを探索している。165 行目では最終的にどこに石を置くのかを出力する。
- CLOSE 相手が切断した場合の例外処理を行っている。

185〜192 行目

例外処理を行っている。

1.4　実行結果

以下が自分のプログラムと MONTEAI を対戦させたときの出力結果である。

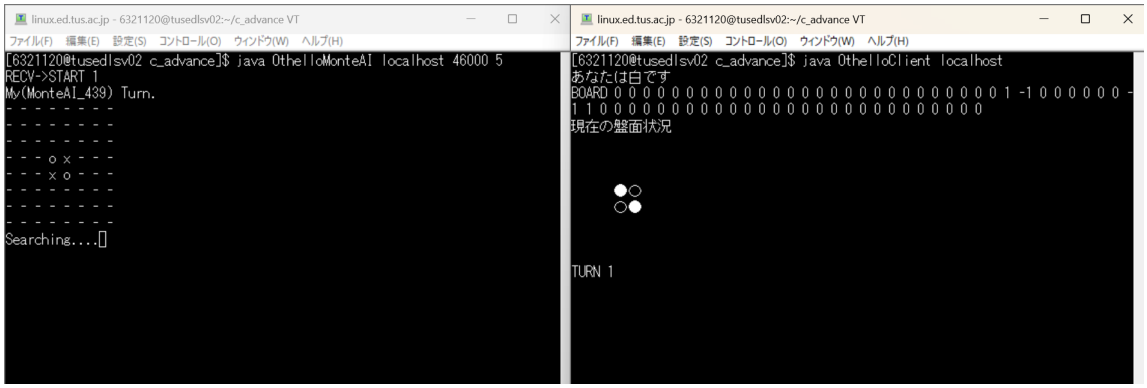

図 1.1　実行結果 1

図 1.2　実行結果 2

1.5 考察

1.5.1 採用したアルゴリズムについて

先述した通り、より強いオセロ AI を作るには置けるマスの中からどのマスに置くかを選ぶアルゴリズムが最重要であると考える。そこで自分が採用した、マスに重み付けを行うだけでは不十分である理由は主に以下であると考えた。

1. 盤面の状況を踏まえて場所を決定できていない。
2. 角を取れば有利という性質に関しては考慮出来ているが、相手の置けるマスを減らすように置くべきというアルゴリズムを実装できていない。

そこで、授業内オセロ大会の優勝者の採用したアルゴリズムの中でαβ法という単語が出てきたため、それについて調べた。

1.5.2 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売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。オセロ AI 大会ではタイムアウト設定は１０秒であったため、この時間内に収める必要がある。この Minimax 探索法の計算量を減らしたものがαβ法である。

1.5.3 αβ法について

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。

1.5.4 重さ付けの値について

自分は１種類の重さ付けを２次元配列に格納して評価材料とした。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下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。

参考文献

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