プログラミングおよび実習 I・第 11 回

2003 年 6 月 25 日

1　コンピューター対戦を実現する

現在のオセロゲームのプログラムは、単にユーザーの選んだ差し手に従ってゲームを進めて行くものでしかありません。これをを改良して、コンピューターとの対戦ができるようなものにしましょう。

適当な方法で選んだ手を差すような関数 autoMove をプログラムに追加し、ユーザーがキーボードから t (あるいは T) の 1 文字を入力した場合に、(関数 inputMove から) この autoMove を呼び出すことにします。この部分のプログラムはつぎのようなものとなります[1]。

```
* Boolean autoMove(OState *state)
* {
      /* 何らかの方法で差し手を選んで putDisk を呼び出し TRUE を返す。
         手がない場合は pass を呼び出し FALSE を返す */
       ⋮
* }

  /* ユーザーが指定した位置に石を置く */
  Boolean inputMove(OState *state)
  {
      char buf[100], ch1, ch2, ch3;
      int n;

      /* 位置を入力する */
      while (TRUE) {
           ⋮
          ch1 = tolower(ch1);
          ch2 = tolower(ch2);
          if (n == 1) {
              /* q の1文字なら 1 を返り値として return する */
              if (ch1 == 'q')
                  return TRUE;
              /* t の1文字ならコンピュータに手を考えさせる */
*             if (ch1 == 't') {
*                 if (autoMove(state))
*                     break;
*                 continue;
*             }
              /* p の1文字なら手番をパスする */
              if (ch1 == 'p') {
                  if (!pass(state))
                      printf("パスはできません\n");
                  continue;
              }
          }
          ⋮
      }
      return FALSE;
  }
```

関数 autoMove は、相手の石を反転できるような差し手の内の 1 つを選んで関数 putDisk を呼び出した後、返り値として TRUE を返すものとします。もし、相手を反転できる手がなかった場合には、関数

[1] 変更あるいは追加された部分の行に * 印を付けてあります

pass を呼び出すことでパスを選択し、autoMove は FALSE を返すようにします。関数 inputMove では、autoMove の返り値を調べて、関数 main に戻って盤面の表示を行う (差し手があった場合) か、あるいはユーザーに対するプロンプトを再度出力する (パスした場合) かを決定します。

1.1 最初に発見した手を選ぶ

コンピューターを強いオセロプレーヤーに仕立てるには、それなりに本格的な方法を使って差し手を選ぶようにしなければなりませんが、単にルール通りに差せばいいのなら、簡単に差し手の選択を実現することができます。まず手始めに、何でもよいから相手の石を反転することのできるような手を探して、最初に見つかった手を差すようにしてみましょう。

盤上のすべてのマスに対して手当たりしだい石を置いてみて、相手の石を反転することができたらそれで満足することにします。この方法を用いた関数 autoMove の定義は次のようなものとなります。

```
Boolean autoMove(OState *state)
{
    int x, y;

    FOR_EACH_POS(x, y) {
        if (putDisk(state, x, y, TRUE) > 0)
            return TRUE;
    }
    pass(state);
    return FALSE;
}
```

1.2 相手の石を最も多く反転させる手を選ぶ

もう少し複雑な差し手の選び方として、相手の石を最も多く反転させるような手を選ぶようにしてみます。あるマスに石を置いて反転する石の数は、最後の引数を FALSE として関数 putDisk を呼び出すことで調べることができますので、これを利用して手を選ぶことができます。これと似たことは、関数 pass の中で、相手の石を反転するような手がないことを確認するときにも行っていました。

```
Boolean autoMove(OState *state)
{
    int c, maxCount = 0;
    int x, y, maxX = -1, maxY = -1;

    /* 相手の石を最も多く反転させるような手を選んで差す。
       手がなければパスする */
    FOR_EACH_POS(x, y) {
        c = putDisk(state, x, y, FALSE);
        /* これまでになく多ければ、その数とその手を記録しておく */
        if (c > maxCount) {
            maxX = x;
            maxY = y;
            maxCount = c;
        }
    }
    if (maxCount > 0) {
        putDisk(state, maxX, maxY, TRUE);
        return TRUE;
    }
    pass(state);
```

```
    return FALSE;
}
```

1.3 相手の手番での差し手の数が最も少なくなるような手を選ぶ

つぎに、相手が選べる手の数が最も少なくなるような差し手を選んでみましょう。以下のようなプログラムでこれを実現することができます。

```
Boolean autoMove(OState *state)
{
    int n, minNum = BOARD_SIZE*BOARD_SIZE;
    int x, y, xx, yy, minX = -1, minY = -1;
    OState next;        /* 1手先の局面 */

    /* 相手の差し手の数が最も少なくなるような手を選んで差す */
    FOR_EACH_POS(x, y) {
        /* まず、現在局面を next にコピーする */
        next = *state;
        /* (x, y) に置いてみて、置けなければ次のマスへ進む */
        if (putDisk(&next, x, y, TRUE) == 0)
            continue;
        /* 相手の差し手の総数を数える */
        n = 0;
        FOR_EACH_POS(xx, yy) {
            if (putDisk(&next, xx, yy, FALSE) > 0)
                n++;
        }
        /* これまでになく少なければ、その数とその手を記録しておく */
        if (n < minNum) {
            minX = x;
            minY = y;
            minNum = n;
        }
    }
    if (minNum < BOARD_SIZE*BOARD_SIZE) {
        putDisk(state, minX, minY, TRUE);
        return TRUE;
    }
    pass(state);
    return FALSE;
}
```

この選択方法では、自分が 1 つの手を差した後の局面 (1 手先の局面) で、相手の差し手がいくつあるのかを数えなければなりません。本当にそこに石を置いて (次の局面に移行して) しまうと元の局面に戻すのが大変ですから、まず state が指している現在の局面を OState型の変数 next にコピーして、この next に石を置いて次の局面を作ってみています。

2 演習問題

2.1 最後に打った石が分かるような表示にする

まず、前回の演習問題で作成したプログラム othello23.c を othello24.c にコピーしなさい。次に、この othello24.c を変更して、以下の実行例のように、最後に打った石が「★」や「☆」で表示されるようにしなさい。

まず、局面の状態を表現している構造体 OState 型に次の 2 つのメンバ lastX と lastY を追加しましょう。

```
int lastX, lastY;   /* 最後に打った石の位置 */
```

関数 reset で局面が初期化されるときに、この 2 つのメンバをともに −1 で初期化しておきます。また、関数 putDisk で石が置かれるときに、その石の位置 (x, y) を lastX と lastY に格納するようにします。後は、関数 showBoard が (lastX, lastY) の位置の石を「★」や「☆」で表示するように変更すれば完成です。

実行例

```
s1609h017% ./othello24
 a b c d
1□□□□
2□○●□
3□●○□
4□□□□
黒番（黒2/白2）= a2
 a b c d
1□□□□
2★●●□
3□●○□
4□□□□
白番（黒4/白1）= a3
 a b c d
1□□□□
2●●●□
3☆○○□
4□□□□
黒番（黒3/白3）= a4
 a b c d
1□□□□
2●●●□
3●●○□
4★□□□
白番（黒6/白1）= c1
 a b c d
1□□☆□
2●●○□
3●●○□
4●□□□
```

続き

```
黒番（黒5/白3）= d3
 a b c d
1□□○□
2●●○□
3●●●★
4●□□□
白番（黒7/白2）= c4
 a b c d
1□□○□
2●●○□
3●●○●
4●□☆□
黒番（黒6/白4）= d2
 a b c d
1□□○□
2●●●★
3●●○●
4●□○□
白番（黒8/白3）= a1
 a b c d
1☆□○□
2●○●●
3●●○●
4●□○□
黒番（黒7/白5）= b1
 a b c d
1○★○□
2●●●●
3●●○●
4●□○□
```

続き

```
白番（黒9/白4）= p
黒番（黒9/白4）= d1
 a b c d
1○●●★
2●●●●
3●●○●
4●□○□
白番（黒11/白3）= p
黒番（黒11/白3）= d4
 a b c d
1○●●●
2●●●●
3●●●●
4●□○★
白番（黒13/白2）= p
黒番（黒13/白2）= b4
 a b c d
1○●●●
2●●●●
3●●●●
4●★●●
黒の勝ちです（黒15/白1）
s1609h017%
```

完成したら「mprog1 othello24.c」のようにして mprog1 コマンドを実行し、正しくできているかどうかチェックしてください。正しくできている場合は、そのプログラムが提出されたことになります。プログラムは Prog1 というディレクトリに作ることを忘れないようにしましょう。次の演習問題も同様です。

2.2 自動的に手を選べるようにする

まず、プログラム othello24.c を othello25.c にコピーしなさい。次に、この othello25.c を変更して、第 1節で解説したいずれかの方法を使って、キーボードから t (あるいは T) の 1 文字を入力すると、プログラムが自動的に差し手を選んで石を置く (あるいはパスする) ようにしなさい。以下は othello25.c の実行例です。

実行例

```
s1609h017% ./othello25
 a b c d
1□□□□
2□○●□
3□●○□
4□□□□
黒番（黒2/白2）= a2
 a b c d
1□□□□
2★●●□
3□●○□
4□□□□
白番（黒4/白1）= c1
 a b c d
1□□☆□
2●●○□
3□●○□
4□□□□
黒番（黒3/白3）= t
 a b c d
1□□○★
2●●●□
3□●○□
4□□□□
```

続き

```
白番（黒5/白2）= t
 a b c d
1□□○●
2●○●□
3☆○○□
4□□□□
黒番（黒3/白5）= T
 a b c d
1□□○●
2●○●□
3●●○□
4★□□□
白番（黒6/白3）= T
 a b c d
1□□○●
2●○●□
3●○○□
4●☆□□
黒番（黒5/白5）= t
 a b c d
1□□○●
2●○●□
3●●●□
4●●★□
```

続き

```
白番（黒9/白2）= 2D
 a b c d
1□□○●
2●○○☆
3●●●□
4●●●□
黒番（黒8/白4）= t
 a b c d
1□★●●
2●●○○
3●●●□
4●●●□
白番（黒11/白2）= t
黒番（黒11/白2）= t
 a b c d
1□●●●
2●●●●
3●●●★
4●●●□
白番（黒14/白0）= t
黒番（黒14/白0）= t
黒の勝ちです（黒14/白0）
s1609h017%
```

余裕のある人は、プログラムが差す手がより良い手となるような工夫をしてみましょう。たとえば、盤の隅に置ける場合はそこを優先するとか、盤の隅に隣接したマスにはできるだけ置かないようにするとか、いろいろ考えられます。ゲームの序盤、中盤、終盤で、差し手の選択方法を切り替えるのも 1 つの方法でしょう。盤の大きさが 4×4 ではものたりない人は、8×8 に戻して試してみましょう。ただし、`mprog1` コマンドを実行して課題を提出する際の盤の大きさは 4×4 にしてください。

次回の予定

次回は `othello25.c` をさらに発展させて、次のような機能を追加する予定です。

(1) オセロゲームのプログラムを「`./othello26 -b`」や「`./othello26 -w`」のようにオプションを付けて起動すると、それぞれ、黒番や白番の手をプログラムが勝手に選んで打ってくれるようにします。

(2) スタックと呼ばれるデータ構造を利用して、プレーヤーが「待った」をできるようにします。

プログラミングおよび実習 I・第 11 回・終り

付録: 前回の演習問題 othello23.c のプログラム例

関数 getLine、showBoard、main の定義は省略してあります。また、othello21.c から変更された行には * 印が付してあります。

othello23.c

```
   1 #include <stdio.h>
   2 #include <ctype.h>
   3
   4 typedef int Boolean;
   5
   6 #define TRUE            (1)
   7 #define FALSE           (0)
   8
   9 #define BLACK           (1)         /* 黒石を表す定数 */
  10 #define WHITE           (-BLACK)    /* 白石を表す定数 */
  11 #define NONE            (0)         /* 石がないことを表す */
  12 #define OPPONENT(t)     (-(t))      /* t の相手 */
* 13 #define BOARD_SIZE      (4)         /* オセロボードの縦横のマス数 */
  14
  15 #define IS_VALID(x)             (0 <= (x) && (x) < BOARD_SIZE)
  16 #define IS_VALID_POS(x, y)      (IS_VALID(x) && IS_VALID(y))
  17 #define FOR_EACH(x)             for ((x) = 0; (x) < BOARD_SIZE; (x)++)
  18 #define FOR_EACH_POS(x, y)      FOR_EACH(x) FOR_EACH(y)
  19
  20 typedef struct {
  21     int board[BOARD_SIZE][BOARD_SIZE];  /* 盤面の状況 */
  22     int turn;                           /* 次の手番 */
* 23     int blackCount;                     /* 黒石の数 */
* 24     int whiteCount;                     /* 白石の数 */
* 25     Boolean passed;                     /* パスした直後か? */
  26 } OState;

   ⋮

  49 /* 盤面を初期化する */
  50 void reset(OState *state)
  51 {
  52     int x, y;
  53
  54     /* 先手は黒 */
  55     state->turn = BLACK;
  56     /* すべてのマスをクリアする */
  57     FOR_EACH_POS (x, y)
  58         state->board[x][y] = NONE;
  59     /* 中央に4つの石を置く */
  60     state->board[BOARD_SIZE/2-1][BOARD_SIZE/2-1] = WHITE;
  61     state->board[BOARD_SIZE/2-1][BOARD_SIZE/2] = BLACK;
  62     state->board[BOARD_SIZE/2][BOARD_SIZE/2-1] = BLACK;
  63     state->board[BOARD_SIZE/2][BOARD_SIZE/2] = WHITE;
  64     /* blackCount と whiteCount を初期化する */
* 65     state->blackCount = state->whiteCount = 2;
  66     /* パスフラグをクリア */
* 67     state->passed = FALSE;
  68 }
  69
  70 /* 勝負けを表示する */
* 71 void showResult(OState *state)
* 72 {
```

```
* 73     if (state->blackCount > state->whiteCount)
* 74         printf("黒の勝ちです");
* 75     else if (state->whiteCount > state->blackCount)
* 76         printf("白の勝ちです");
* 77     else
* 78         printf("引き分けです");
* 79     printf(" (黒%d/白%d)\n", state->blackCount, state->whiteCount);
* 80 }

   ⋮

 140 /* (x, y) の位置に手番の石を置いたとして、挟まれる相手の石の数を返り
 141    値として返す。update が真で、かつ挟まれた石があれば実際にそこに石を
 142    置き手番を交代する。 */
 143 int putDisk(OState *state, int x, int y, Boolean update)
 144 {
 145     int count = 0;      /* 挟まれた石の数 */
 146     int dx, dy;
 147
 148     /* 盤の外や、すでに石のあるマスなら石を置かずに 0 を返す */
 149     if (!IS_VALID_POS(x, y) || state->board[x][y] != NONE)
 150         return 0;
 151     for (dx = -1; dx <= 1; dx++)
 152         for (dy = -1; dy <= 1; dy++)
 153             if (dx != 0 || dy != 0)
 154                 count += checkDir(state, x, y, dx, dy, update);
 155     /* update が真で、石が挟まれたのなら石を置き手番を変える */
 156     if (update && count > 0) {
*157         if (state->turn == BLACK) {
*158             state->blackCount += count + 1;
*159             state->whiteCount -= count;
*160         }
*161         else {
*162             state->whiteCount += count + 1;
*163             state->blackCount -= count;
*164         }
 165         state->board[x][y] = state->turn;
 166         /* パスフラグを下ろす */
*167         state->passed = FALSE;
 168         /* マスがすべて埋っていたら終局する */
*169         if (state->blackCount+state->whiteCount == BOARD_SIZE*BOARD_SIZE)
*170             state->turn = NONE;
*171         else
*172             state->turn = OPPONENT(state->turn);
 173     }
 174     /* 反転した石の数を返す */
 175     return count;
 176 }
 177
 178 /* 手番をパスして、返り値として 1 を返す。 相手を挟める手がある場合
 179    にはパスをせずに 0 を返す */
 180 Boolean pass(OState *state)
 181 {
 182     int x, y;
 183
 184     FOR_EACH_POS(x, y) {
 185         if (putDisk(state, x, y, FALSE) > 0)
 186             return FALSE;
 187     }
```

```
*188     if (state->passed)
*189         state->turn = NONE;
*190     else
 191         state->turn = OPPONENT(state->turn);
*192     state->passed = TRUE;
 193     return TRUE;
 194 }
 195
 196 /* ユーザーが指定した位置に石を置く */
 197 Boolean inputMove(OState *state)
 198 {
 199     char buf[100], ch1, ch2, ch3;
 200     int n;
 201
 202     /* 位置を入力する */
 203     while (TRUE) {
 204         /* 終局していたら勝ち負けを表示してプログラムを終了する */
*205         if (state->turn == NONE) {
*206             showResult(state);
*207             return TRUE;
*208         }
*209         printf("%s番 (黒%d/白%d) = ", (state->turn == BLACK) ? "黒" : "白",
*210                state->blackCount, state->whiteCount);
 211         /* キーボードからの1行分の入力を buf に格納する */
 212         if (getLine(buf, sizeof buf) < 1)
 213             continue;
 214         /* buf から空白類を無視して、最大3文字を抜き出す */
 215         n = sscanf(buf, " %c %c %c", &ch1, &ch2, &ch3);
 216         ch1 = tolower(ch1);
 217         ch2 = tolower(ch2);
 218         if (n == 1) {
 219             /* q の1文字なら 1 を返り値として return する */
 220             if (ch1 == 'q')
 221                 return TRUE;
 222             /* p の1文字なら手番をパスする */
 223             if (ch1 == 'p') {
 224                 if (!pass(state))
 225                     printf("パスはできません\n");
 226                 continue;
 227             }
 228         }
 229         /* 2文字でないなら再入力を促す */
 230         if (n != 2)
 231             continue;
 232         /* とりあえず c5 のような形式の入力と仮定してみる */
 233         if (putDisk(state, ch1 - 'a', ch2 - '1', TRUE) > 0)
 234             break;
 235         /* 逆に 5c のような形式の入力と仮定してみる */
 236         if (putDisk(state, ch2 - 'a', ch1 - '1', TRUE) > 0)
 237             break;
 238         printf("そこには置けません\n");
 239     }
 240     return FALSE;
 241 }
```

⋮