

—

2022 年度 ソフトウェア技術

C 言語 [演習]

2023 年 10 月 8 日

S.Matoi

目次

第 1 章	三目並べ (Tic Tac Toe)	3
1.1	ゲームの概要	3
1.2	主処理 (main)	4
1.3	盤面の表示 (printBoard)	4
1.4	手番の交代 (switchTurn)	4
1.5	入力 (slotNum)	5
1.6	判定 (checkWinner)	5
1.7	結果表示 (result)	6
1.8	各種宣言など	6
第 2 章	スライド・パズル (15Puzzle)	7
2.1	ゲームの概要	7
2.2	主処理 (main)	8
2.3	盤面の初期化 (init)	8
2.4	シャッフル (shuffle)	8
2.5	盤面の表示 (disp)	9
2.6	タイルのスライド (moveTile)	9
2.7	完成チェック (check)	10
2.8	各種宣言など	10
第 3 章	神経衰弱 (Flip Cards)	12
3.1	ゲームの概要	12
3.2	主処理 (main)	14
3.3	盤面の初期化 (init)	14
3.4	カードのシャッフル (shuffle, swapCards)	15
3.5	盤面の表示 (disp)	15
3.6	入力 (getNum)	15
3.7	カードの開閉 (openCard, closeCard)	16
3.8	一致不一致の判定 (match)	16
3.9	各種宣言など	16
第 4 章	オセロ (リバーシ)	17
4.1	ゲームの概要	18

目次	2
4.2 各種宣言など	19
4.3 主処理	20
4.4 初期化	21
4.5 盤面の表示	21
4.6 手番の交代	22
付録 A 全体プログラム	24
A.1 三目並べ	24
A.2 スライド・パズル	27
A.3 神経衰弱	30
A.4 オセロ（リバーシ）	33
参考文献	37

第 1 章

三目並べ (Tic Tac Toe)

1.1 ゲームの概要

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、先に一行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することはできません。また、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 | 0 |
|---|---|---|
| 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 主処理 (main)

```
1 int main(int argc, char *argv[]){
2     /* 先手後手を決定 */
3     int turn = BATSU, winner, num;
4     if (1 < argc){
5         if (!strcmp(argv[1], "-r"))
6             turn = MARU;
7     }
8     printf("スタート!_Tic_Tac_Toe\n");
9     do{
10        printBoard();          /* ① 盤面の表示 */
11        num = slotNum(turn); /* ② 手を入力 */
12        board[num] = turn;    /* ③ 手を盤面に配置 */
13        turn = switchTurn(turn); /* ④ 手番の交代 */
14        winner = checkWinner(); /* ⑤ 勝敗の判定 */
15    } while (winner == NEXT);
16    /* 対戦結果の表示 */
17    printBoard();
18    result(winner);
19    return 0;
20 }
```

1.3 盤面の表示 (printBoard)

```
1 void printBoard() {
2     char bd[9];
3     int i;
4     for (i = 0; i < 9; i++) {
5         if (board[i] == MARU) bd[i] = 'O';
6         else if (board[i] == BATSU) bd[i] = 'X';
7         else bd[i] = '0' + i;
8     }
9     printf("\n/---|---|---\\n");
10    printf("|_c_|_c_|_c_|_c_|", bd[0], bd[1], bd[2]);
11    printf("|---|---|---|n");
12    printf("|_c_|_c_|_c_|_c_|", bd[3], bd[4], bd[5]);
13    printf("|---|---|---|n");
14    printf("|_c_|_c_|_c_|_c_|", bd[6], bd[7], bd[8]);
15    printf("\\---|---|---/n");
16 }
```

1.4 手番の交代 (switchTurn)

```
1 int switchTurn(int turn) {
2     if (turn== BATSU) return MARU;
3     return BATSU;
4 }
```

1.5 入力 (slotNum)

```
1 int slotNum(int turn) {
2     int num;
3     char *fig = "";
4     if (turn==MARU) fig = "'O'";
5     else if (turn==BATSU) fig = "'X'";
6     do {
7         printf("\n%sさんのturnです\n石を置く場所 0~8を指定して下さい:", fig);
8         //while (getchar() != '\n'); /* 標準入力バッファのクリア */
9         scanf("%d", &num);
10        if (!(0 <= num && num < 9)) {
11            printf("再指定: 0~8を指定して下さい");
12            continue;
13        }
14        if (board[num] != num) {
15            printf("再指定: そこには既に石が置かれています\n");
16            continue;
17        }
18        break;
19    } while (1);
20    return num;
21 }
```

1.6 判定 (checkWinner)

```
1 int lineSum(int n1, int n2, int n3) {
2     return board[n1] + board[n2] + board[n3];
3 }
4 int checkWinner() {
5     int i, line = 0;
6     for (i = 0; i < 8; i++) {
7         switch (i) {
8             case 0: line = lineSum(0, 1, 2); break;
9             case 1: line = lineSum(3, 4, 5); break;
10            case 2: line = lineSum(6, 7, 8); break;
11            case 3: line = lineSum(0, 3, 6); break;
12            case 4: line = lineSum(1, 4, 7); break;
13            case 5: line = lineSum(2, 5, 8); break;
14            case 6: line = lineSum(0, 4, 8); break;
15            case 7: line = lineSum(2, 4, 6); break;
16        }
17        if (line == 3 * MARU) return MARU;
18        else if (line == 3 * BATSU) return BATSU;
19    }
20    for (i = 0; i < 9; i++){
21        if (0 <= board[i] && board[i] < 9) return NEXT;
22    }
23    return DRAW;
24 }
```

1.7 結果表示 (result)

```
1 void result(int winner) {
2     printf("\n");
3     switch (winner) {
4         case DRAW: printf("引き分け\t"); break;
5         case MARU: printf("'O'の勝ち\t"); break;
6         case BATSU: printf("'X'の勝ち\t"); break;
7     }
8     printf("またね!\n");
9 }
```

1.8 各種宣言など

これは冒頭に記述する

```
1 #include <stdio.h>
2 #include <string.h>
3
4 // function prototypes
5 /*
6  int lineSum(int, int, int);
7  int switchTurn(int);
8  void printBoard();
9  int slotNum(int);
10 int checkWinner();
11 void result(int);
12 */
13
14 static int board[] = {0, 1, 2, 3, 4, 5, 6, 7, 8};
15 #define MARU 10
16 #define BATSU -10
17 #define DRAW 100
18 #define NEXT 200
```

第2章

スライド・パズル (15Puzzle)

2.1 ゲームの概要

4×4 に区切った盤面上の各タイルに、0～15 の番号が割り振られている。0 が割り振られたタイルは空欄になっていて、他のタイルはその空欄にスライドさせて移動することができる。最初は不規則に並べられている盤面ですが、空欄の上下、あるいは空欄の左右のタイルを選んで、空欄の方向にスライドさせる事によって、最終的に 1 から 15 まで規則正しく並んだ盤面状態を目指すゲームである。

```
% ./slidetile
```

```
[11][ 3][ 6][10]
[ 9][ 1][ 8][14]
[ 2][ 4][ 7][12]
[13][15][ ][ 5]
```

```
Select number:6
```

```
[11][ 3][ ][10]
[ 9][ 1][ 6][14]
[ 2][ 4][ 8][12]
[13][15][ 7][ 5]
```

```
Select number:11
```

```
[ ][11][ 3][10]
[ 9][ 1][ 6][14]
[ 2][ 4][ 8][12]
[13][15][ 7][ 5]
```

```
Select number:11
```

```
[11][ ][ 3][10]
[ 9][ 1][ 6][14]
[ 2][ 4][ 8][12]
[13][15][ 7][ 5]
```

```
Select number:
```


2.2 主処理 (main)

```
1 int main(void) {
2     int sel;
3     init();
4     do {
5         disp();
6         do {
7             printf("\nSelect number:");
8             scanf("%d", &sel);
9         } while( sel>ROW*COLUMN );
10        moveTile(sel);
11    } while( check() );
12
13    return 0;
14 }
```

2.3 盤面の初期化 (init)

乱数を使ってタイルをシャフルするので、時刻を乱数の種に指定することによって、プログラムを起動する度に、異なるタイル配置になる様になっている

```
1 void init(){
2     int i, j, n;
3     srand((unsigned)time(NULL));
4     for (i = 0; i < ROW; i++){
5         for (j = 0; j < COLUMN; j++){
6             n = i * COLUMN + j;
7             tiles[n].num = n;
8             tiles[n].row = i;
9             tiles[n].clm = j;
10        }
11    }
12    shuffle(1000);
13 }
```

2.4 シャッフル (shuffle)

乱数を使って、タイルを不規則に選択し移動させている

```
1 void shuffle(int n){
2     int sel, min = 0, max = ROW*COLUMN;
3     do{
4         sel = (rand() % (max - min + 1)) + min;
5         moveTile(sel);
6         n--;
7     } while (0 < n);
8 }
```

2.5 盤面の表示 (disp)

```
1 void disp(){
2     int i, j, n;
3     printf("\n");
4     for (i = 0; i < ROW; i++){
5         for (j = 0; j < COLUMN; j++){
6             n = i * COLUMN + j;
7             if (tiles[n].num == 0){
8                 printf("[  ]");
9             } else {
10                printf("[%2d]", tiles[i * COLUMN + j].num);
11            }
12        }
13        printf("\n");
14    }
15 }
```

2.6 タイルのスライド (moveTile)

```
1 int findTileNum(int num){
2     int i;
3     for (i = 0; i < ROW * COLUMN; i++){
4         if (tiles[i].num == num){
5             return i;
6         }
7     }
8     return -1;
9 }
10 void swapTile(int n1, int n2){
11     int tmp = tiles[n1].num;
12     tiles[n1].num = tiles[n2].num;
13     tiles[n2].num = tmp;
14 }
15 void swapTileR(int c, int r1, int r2){
16     int n1 = r1 * COLUMN + c;
17     int n2 = r2 * COLUMN + c;
18     swapTile(n1, n2);
19 }
20 void swapTileC(int r, int c1, int c2){
21     int n1 = r * COLUMN + c1;
22     int n2 = r * COLUMN + c2;
23     swapTile(n1, n2);
24 }
25 void moveTile(int sel){
26     int j;
27     int s = findTileNum(sel);
28     int sr = tiles[s].row;
29     int sc = tiles[s].clm;
30     int z = findTileNum(0);
31     int zr = tiles[z].row;
32     int zc = tiles[z].clm;
33     if (sr == zr){
34         if (sc < zc){
35             for (j = zc; j > sc; j--){
36                 if (0 <= j - 1){
37                     swapTileC(sr, j, j - 1);
38                 }

```

```

39     }
40     } else if (zc < sc) {
41         for (j = zc; j < sc; j++){
42             if (j + 1 < COLUMN){
43                 swapTileC(sr, j, j + 1);
44             }
45         }
46     }
47 }
48 if (sc == zc){
49     if (sr < zr){
50         for (j = zr; j > sr; j--){
51             if (0 <= j - 1){
52                 swapTileR(sc, j, j - 1);
53             }
54         }
55     } else if (zr < sr){
56         for (j = zr; j < sr; j++){
57             if (j + 1 < ROW){
58                 swapTileR(sc, j, j + 1);
59             }
60         }
61     }
62 }
63 }

```

2.7 完成チェック (check)

```

1  enum BOOLEAN check(){
2      int i, j, n;
3      for (i = 0; i < ROW; i++){
4          for (j = 0; j < COLUMN; j++){
5              n = i * COLUMN + j;
6              if (tiles[n].num != n){
7                  return true;
8              }
9          }
10     }
11     return false;
12 }

```

2.8 各種宣言など

これは冒頭に記述する

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4
5  // 定数
6  #define ROW 4
7  #define COLUMN 4
8  // 構造体
9  struct Tile{
10     int num;
11     int row;
12     int clm;
13 };

```

```
14 enum BOOLEAN{
15     false, /* false = 0, true = 1 */
16     true
17 };
18 // function prototypes
19 /*
20 int findTileNum(int);
21 void swapTile(int,int);
22 void swapTileR(int,int,int);
23 void swapTileC(int,int,int);
24 void moveTile(int);
25 void shuffle(int);
26 enum BOOLEAN check();
27 void disp();
28 void init();
29 */
30
31 static struct Tile tiles[16]; /* 16 = ROW * COLUMN */
```

第3章

神経衰弱 (Flip Cards)

3.1 ゲームの概要

カードを2枚開いて、一致すれば得点となり、不一致なら再度伏せて相手の手番になる。

```
% ./flipcard
[ a ][ b ][ c ][ d ][ e ]
[ f ][ g ][ h ][ i ][ j ]
[ k ][ l ][ m ][ n ][ o ]
[ p ][ q ][ r ][ s ][ t ]

A さん：0 点 B さん：0 点    = A さんの番です =
Select 1st card : a

[ 4 ][ b ][ c ][ d ][ e ]
[ f ][ g ][ h ][ i ][ j ]
[ k ][ l ][ m ][ n ][ o ]
[ p ][ q ][ r ][ s ][ t ]

Select 2nd card : p

[ 4 ][ b ][ c ][ d ][ e ]
[ f ][ g ][ h ][ i ][ j ]
[ k ][ l ][ m ][ n ][ o ]
[ 5 ][ q ][ r ][ s ][ t ]

ハズレ

[ a ][ b ][ c ][ d ][ e ]
[ f ][ g ][ h ][ i ][ j ]
[ k ][ l ][ m ][ n ][ o ]
[ p ][ q ][ r ][ s ][ t ]

A さん：0 点 B さん：0 点    = B さんの番です =
Select 1st card : e

[ a ][ b ][ c ][ d ][ 4 ]
[ f ][ g ][ h ][ i ][ j ]
[ k ][ l ][ m ][ n ][ o ]
[ p ][ q ][ r ][ s ][ t ]
```

Select 2nd card : a

```
[ 4 ][ b ][ c ][ d ][ 4 ]
[ f ][ g ][ h ][ i ][ j ]
[ k ][ l ][ m ][ n ][ o ]
[ p ][ q ][ r ][ s ][ t ]
```

当たり

```
[ 4 ][ b ][ c ][ d ][ 4 ]
[ f ][ g ][ h ][ i ][ j ]
[ k ][ l ][ m ][ n ][ o ]
[ p ][ q ][ r ][ s ][ t ]
```

A さん : 0 点 B さん : 1 点 = B さんの番です =

Select 1st card : t

```
[ 4 ][ b ][ c ][ d ][ 4 ]
[ f ][ g ][ h ][ i ][ j ]
[ k ][ l ][ m ][ n ][ o ]
[ p ][ q ][ r ][ s ][ 0 ]
```

Select 2nd card : q

```
[ 4 ][ b ][ c ][ d ][ 4 ]
[ f ][ g ][ h ][ i ][ j ]
[ k ][ l ][ m ][ n ][ o ]
[ p ][ 9 ][ r ][ s ][ 0 ]
```

ハズレ

```
[ 4 ][ b ][ c ][ d ][ 4 ]
[ f ][ g ][ h ][ i ][ j ]
[ k ][ l ][ m ][ n ][ o ]
[ p ][ q ][ r ][ s ][ t ]
```

A さん : 0 点 B さん : 1 点 = A さんの番です =

Select 1st card :

3.2 主処理 (main)

```
1 int main( void ){
2     int cnum1, cnum2, Closed=ROW*COLUMN;
3     int pointA=0, pointB=0;
4     init();
5     disp();
6     do{
7         printf("Aさん：%d点\tBさん：%d点\n", pointA, pointB);
8         if (Turn)
9             printf("= Aさんの番です\n");
10        else
11            printf("= Bさんの番です\n");
12        cnum1 = getNum("1st");
13        cnum2 = getNum("2nd");
14        if ( match(cnum1, cnum2) ){
15            printf("当たり\n");
16            Closed -= 2;
17            if (Turn)
18                pointA++;
19            else
20                pointB++;
21        } else {
22            printf("ハズレ\n");
23            closeCard(cnum1);
24            closeCard(cnum2);
25            Turn = !Turn;    /* 手番の交代 */
26        }
27        sleep(4);    /* 開いたカードを見せておく時間 */
28        disp();
29    } while ( 0 < Closed );
30
31    return 0;
32 }
```

3.3 盤面の初期化 (init)

プログラム中で乱数を使うので、時刻を乱数の種に指定して、プログラムを起動するたびに異なるカード配置になるようにしている

```
1 void init(){
2     srand((unsigned)time(NULL));
3     int i, j, num;
4     for (i = 0; i < ROW/2; i++){
5         for (j = 0; j < COLUMN; j++){
6             num = i * COLUMN + j;
7             cards[num].num = cards[ROW * COLUMN - num].num = num;
8             cards[num].row = cards[ROW * COLUMN - num].row = i;
9             cards[num].clm = cards[ROW * COLUMN - num].clm = j;
10            cards[num].opn = cards[ROW * COLUMN - num].opn = false;
11        }
12    }
13    shuffle();
14 }
```

3.4 カードのシャフル (shuffle, swapCards)

乱数を使ってカード配置を入れ替えている

```
1 void swapCards(int n1, int n2){
2     struct Card temp;
3     temp = cards[n2];
4     cards[n2] = cards[n1];
5     cards[n1] = temp;
6 }
7 void shuffle(){
8     int sel, min = 0, max = ROW*COLUMN;
9     while (0 < max){
10         sel = (rand() % (max - min + 1)) + min;
11         swapCards(sel, max--);
12     }
13 }
```

3.5 盤面の表示 (disp)

```
1 void disp(){
2     int i, j, n;
3     printf("\n");
4     for (i = 0; i < ROW; i++){
5         for (j = 0; j < COLUMN; j++){
6             n = i * COLUMN + j;
7             if ( !cards[n].opn )
8                 printf("[_%c_]", 'a' + i * COLUMN + j );
9             else
10                 printf("[_%d_]", cards[i * COLUMN + j].num );
11         }
12         printf("\n");
13     }
14     printf("\n");
15 }
```

3.6 入力 (getNum)

```
1 int getNum(char* s){
2     int cnum;
3     char str[24], work[24];
4     do {
5         sprintf(work, "%s%s%s", "\tSelect_", s, "_card:_");
6         printf("%s", work);
7         scanf("%s", str);
8         cnum = str[0] - 'a';
9     } while (cnum > ROW * COLUMN);
10    openCard(cnum);
11    disp();
12    return cnum;
13 }
```


3.7 カードの開閉 (openCard, closeCard)

```
1 void openCard(int n){
2     cards[n].opn = true;
3 }
4 void closeCard(int n){
5     cards[n].opn = false;
6 }
```

3.8 一致不一致の判定 (match)

```
1 enum BOOLEAN match(int n1, int n2){
2     if ( cards[n1].num == cards[n2].num )
3         return true;
4     return false;
5 }
```

3.9 各種宣言など

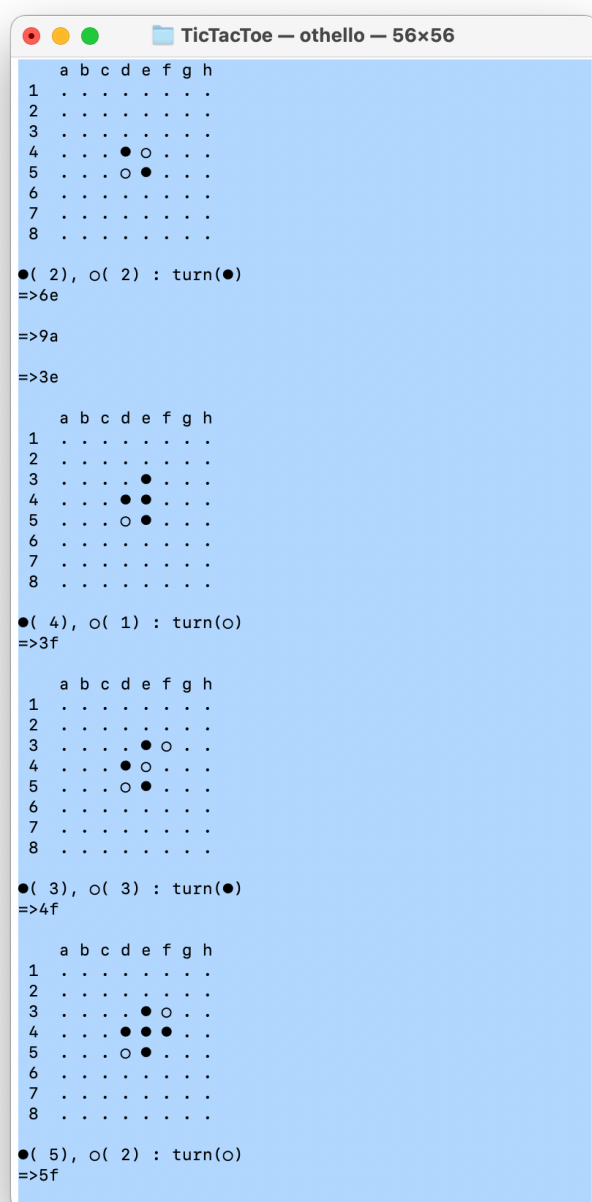
これは冒頭に記述する

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4 #include <unistd.h>
5 // 定数
6 #define ROW 4
7 #define COLUMN 5
8 // 構造体
9 enum BOOLEAN {
10     false, /* false = 0, true = 1 */
11     true
12 };
13 struct Card {
14     int num;
15     int row;
16     int clm;
17     enum BOOLEAN opn;
18 };
19 // function prototypes
20 /*
21 void openCard(int);
22 void closeCard(int);
23 void switchTurn(enum BOOLEAN);
24 void disp();
25 int getNum(char*);
26 enum BOOLEAN match(int, int);
27 void wasteTime(int);
28 void swapCards(int, int);
29 void shuffle();
30 void init();
31 */
32 static struct Card cards[20]; /* 20 = ROW * COLUMN */
33 enum BOOLEAN Turn = true;
```


第4章

オセロ（リバーシ）

4.1 ゲームの概要



```
TicTacToe — othello — 56x56
a b c d e f g h
1 . . . . . . . .
2 . . . . . . . .
3 . . . . . . . .
4 . . . ● ○ . . .
5 . . . ○ ● . . .
6 . . . . . . . .
7 . . . . . . . .
8 . . . . . . . .

●( 2), ○( 2) : turn(●)
=>6e

=>9a

=>3e

a b c d e f g h
1 . . . . . . . .
2 . . . . . . . .
3 . . . ● . . . .
4 . . . ● ● . . .
5 . . . ○ ● . . .
6 . . . . . . . .
7 . . . . . . . .
8 . . . . . . . .

●( 4), ○( 1) : turn(○)
=>3f

a b c d e f g h
1 . . . . . . . .
2 . . . . . . . .
3 . . . ● ○ . . .
4 . . . ● ○ . . .
5 . . . ○ ● . . .
6 . . . . . . . .
7 . . . . . . . .
8 . . . . . . . .

●( 3), ○( 3) : turn(●)
=>4f

a b c d e f g h
1 . . . . . . . .
2 . . . . . . . .
3 . . . ● ○ . . .
4 . . . ● ● ● . .
5 . . . ○ ● . . .
6 . . . . . . . .
7 . . . . . . . .
8 . . . . . . . .

●( 5), ○( 2) : turn(○)
=>5f
```

4.2 各種宣言など

1. これは冒頭に記述する
2. BOOLEAN 型を enum で定義する
3. 盤面の幅 BOARDW は #define 文で指定する
4. 盤面に黒の石が置かれている場所には BLACK、白の石が置かれている場所は WHITE、何も置かれていない場所は NONE で区別する
5. 現在注目している石の位置 (x,y) から見て、下の方向には (0,-1)、右下には (1,-1)、右には (0,1)、右上には (1,1)、上には (1,0)、左上には (-1,1)、左には (-1,0)、左下には (-1,-1) を加えることで、上下、左右、斜め右上、斜め左下の、全部で 8 つの方向の場所を確認していくことができる
6. 盤面上の場所 (x,y) を指定するための構造体 POS を定義する
7. 盤面状態を保持している 1 次元配列 board 上の位置は、POS 型変数 (x,y) から $y \times \text{BOARDW} + x$ によって求める
8. 手番は変数 turn に保持している
9. パスの回数 passcount が 2 回になるとゲームは終了する
10. 終了フラグ endflag が偽である間ゲームは続く
11. setstone() 関数は盤面の指定位置に石を置く操作
12. getstone() 関数は盤面の指定位置の状態を知る操作

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  enum BOOLEAN{
5      false,    /* false=0, true=1 */
6      true
7  };
8
9  #define BOARDW (8) // 4, 6, 8, ....
10 #define BLACK (0)
11 #define WHITE (1)
12 #define NONE (2)
13
14 int UNITV[][2] = {{0,-1},{1,-1},{1,0},{1,1},{0,1},{-1,1},{-1,0},{-1,-1}};
15 int turn = BLACK;
16 int passcount = 0;
17 int endflag = false;
18 int board[BOARDW * BOARDW];
19
20 const char* TILE[] = {
21     "●", //TILE_BLACK
22     "○", //TILE_WHITE
23     ".", //TILE_NONE
24 };
25
26 typedef struct {
27     int x, y;
28 }POS;
29
30 void setstone(POS pos, int num){
31     int index = (pos.y * BOARDW) + pos.x;
32     board[index] = num;
33 }

```

```
34
35     int getstone(POS pos){
36         int index = (pos.y * BOARDW) + pos.x;
37         return board[index];
38     }
```

4.3 主処理

1. 行番号 (1,2,3,...) と列記号文字 (a,b,c,...) の連続する半角 2 文字で石の位置を指定する
2. decode() 関数を使って、入力された位置の文字列を POS 型変数に変換する (ここに 2 桁の行番号は想定していない)
3. 手番の石を置けない場所 (盤面の外: !isinside()、又は反転できる石がない: flippable() がゼロ) を指定しても無視する
4. 現在位置 pos の周辺 8 方向に、それぞれ search() で反転できる石の数を数えて反転していく

```
1     POS decode(char* str){
2         POS pos;
3         pos.x = atoi(str) - 1;
4         pos.y = *(str+1) - 'a';
5         return pos;
6     }
7
8     void event(POS pos){
9         if(endflag){
10             initboard();
11             return;
12         }
13         if(0<passcount){
14             nextturn();
15             drawboard();
16             return;
17         }
18         if(!isinside(pos))
19             return;
20         if(0==flippable(pos, turn))
21             return;
22         for(int i=0; i<8; i++){
23             int loop = search(pos, i, turn);
24             POS temp = pos;
25             for(int j=0; j<loop; j++){
26                 temp = movepos(temp, i);
27                 setstone(temp, turn);
28             }
29         }
30         setstone(pos, turn);
31         nextturn();
32         drawboard();
33     }
34
35     int main(void){
36         if(!initboard())
37             return 8;
38         char inpt []="    ";
39         while(!endflag){
40             printf("=>");
41             scanf("%s", inpt);
42             POS pos = decode(inpt);
43             printf("\n");
```

```

44         event(pos);
45     }
46     return 0;
47 }

```

4.4 初期化

1. ゲームの盤面の幅は、4 以上 8 以下の偶数である

```

1  enum BOOLEAN initboard(){
2      if((BOARDW%2)|| (BOARDW<4)|| (8<BOARDW))
3          return false;
4      int x, y;
5      POS pos;
6      for(y=0; y<BOARDW; y++){
7          for(x=0; x<BOARDW; x++){
8              pos.x = x; pos.y = y;
9              setstone(pos, NONE);
10         }
11         pos.x = pos.y = BOARDW/2-1;
12         setstone(pos, BLACK);
13         pos.x = BOARDW/2; pos.y = pos.x-1;
14         setstone(pos, WHITE);
15         pos.y = BOARDW/2; pos.x = pos.y-1;
16         setstone(pos, WHITE);
17         pos.x = pos.y = BOARDW/2;
18         setstone(pos, BLACK);
19         turn = BLACK;
20         passcount = 0;
21         endflag = false;
22         drawboard();
23         return true;
24     }

```

4.5 盤面の表示

1. 盤面を表示するたびに、count() 関数で石の数を数えて表示している。

```

1  int count(int color){
2      int num=0;
3      for(int i=0; i<(BOARDW * BOARDW); i++){
4          if(board[i]==color)
5              num++;
6      return num;
7  }
8
9  void drawboard(){
10     for(int x=0; x<BOARDW; x++){
11         if(x==0){
12             printf("UUUU");
13             char a='a';
14             for(int i=0; i<BOARDW; i++){
15                 printf("%cU", a+i);
16                 printf("\n");
17             }
18             printf("%2dUU", x+1);
19             for(int y=0; y<BOARDW; y++){

```

```

20         int index = y * BOARDW + x;
21         switch(board[index]){
22             case BLACK:printf("%s□", TILE[BLACK]);break;
23             case WHITE:printf("%s□", TILE[WHITE]);break;
24             case NONE: printf("%s□", TILE[NONE]); break;
25         }
26     }
27     printf("\n");
28 }
29 printf("\n%s(%2d),□s(%2d)",TILE[BLACK],count(BLACK),TILE[WHITE],count(WHITE
30 ));
31 if(!endflag)
32     printf("□:□turn(%s)\n", TILE[turn]);
33 else
34     printf("\n");

```

4.6 手番の交代

1. isinside() 関数は、指定された場所が盤の内部の位置かどうかを調べる
2. flippable() 関数は、相手の石を何枚反転させられるかを調べる
3. movepos() 関数は、着目点を現在位置 pos から上下左右斜めの 8 方向に移動させる
4. search() 関数は、v で指示された方向に何個の石を反転させられるかを数えている
5. nextturn() 関数では、
 石を置ける空いてる場所があるかどうかを調べて無かったら終了フラグを立てる
 指定位置 pos が相手の石を反転できる場所ならパスの回数をリセットする
 パスの回数が 2 回続いたら終了フラグを立てる

```

1  POS movepos(POS pos, int v){
2      POS p;
3      p.x = pos.x + UNITV[v][0];
4      p.y = pos.y + UNITV[v][1];
5      return p;
6  }
7
8  enum BOOLEAN isinside(POS pos){
9      if( (pos.x<0) || (BOARDW<=pos.x) )
10         return false;
11      if( (pos.y<0) || (BOARDW<=pos.y) )
12         return false;
13      return true;
14  }
15
16  int search(POS pos, int v, int num){
17      int piece = 0;
18      while(true){
19          pos = movepos(pos, v);
20          if(!isinside(pos))
21              return 0;
22          if(getstone(pos)==NONE)
23              return 0;
24          if(getstone(pos)==num)
25              break;
26          piece ++;
27      }
28      return piece;
29  }

```

```
30
31     int flippable(POS pos, int num){
32         if(getstone(pos)!=NONE)
33             return 0;
34         int total = 0;
35         int vec[]={0,0};
36         for(int i=0; i<8; i++)
37             total += search(pos, i, num);
38         return total;
39     }
40
41     void nextturn(){
42         turn ^= 1;
43         int empty = 0;
44         for(int y=0; y<BOARDW; y++)
45             for(int x=0; x<BOARDW; x++){
46                 POS pos; pos.x=x; pos.y=y;
47                 if(getstone(pos)==NONE)
48                     empty++;
49                 if(0<flippable(pos,turn)){
50                     passcount = 0;
51                     return;
52                 }
53             }
54         if(empty==0){
55             endflag = true;
56             return;
57         }
58         passcount++;
59         if(2<=passcount)
60             endflag = true;
61     }
```


付録 A

全体プログラム

A.1 三目並べ

ソースコード A.1 Tic Tac Toe(三目並べ)

```
1  #include <stdio.h>
2  #include <string.h>
3
4  // function prototypes
5  /*
6  int lineSum(int, int, int);
7  int switchTurn(int);
8  void printBoard();
9  int slotNum(int);
10 int checkWinner();
11 void result(int);
12 */
13
14 static int board[] = {0, 1, 2, 3, 4, 5, 6, 7, 8};
15 #define MARU 10
16 #define BATSU -10
17 #define DRAW 100
18 #define NEXT 200
19 /* ===== */
20 int switchTurn(int turn) {
21     if ( turn == BATSU) return MARU;
22     return BATSU;
23 }
24 /* ===== */
25 void printBoard() {
26     char bd[9];
27     int i;
28     for (i = 0; i < 9; i++) {
29         if ( board[i] == MARU ) bd[i] = 'O';
30         else if ( board[i] == BATSU ) bd[i] = 'X';
31         else bd[i] = '0' + i;
32     }
33     printf("\n/---|---|---\\n");
34     printf("|_%c|_%c|_%c|\\n", bd[0], bd[1], bd[2]);
35     printf("|---|---|---|\\n");
36     printf("|_%c|_%c|_%c|\\n", bd[3], bd[4], bd[5]);
37     printf("|---|---|---|\\n");
38     printf("|_%c|_%c|_%c|\\n", bd[6], bd[7], bd[8]);
39     printf("\\---|---|---/\\n");
40 }
41 /* ===== */
42 void result(int winner) {
```

```

43     printf("\n");
44     switch (winner) {
45     case DRAW: printf("引き分け\t"); break;
46     case MARU: printf("'O'の勝ち\t"); break;
47     case BATSU: printf("'X'の勝ち\t"); break;
48     }
49     printf("またね!\n");
50 }
51 /* ===== */
52 int slotNum(int turn) {
53     int num;
54     char *fig = "";
55     if (turn == MARU) fig = "'O'";
56     else if (turn == BATSU) fig = "'X'";
57     do {
58         printf("\n%sさんのturnです\n石を置く場所 0~8 を指定して下さい:", fig);
59         //while (getchar() != '\n'); /* 標準入力バッファのクリア */
60         scanf("%d", &num);
61         if (!(0 <= num && num < 9)) {
62             printf("再指定: 0~8 を指定して下さい");
63             continue;
64         }
65         if (board[num] != num) {
66             printf("再指定: そこには既に石が置かれています\n");
67             continue;
68         }
69         break;
70     } while (1);
71     return num;
72 }
73 /* ===== */
74 int lineSum(int n1, int n2, int n3) {
75     return board[n1] + board[n2] + board[n3];
76 }
77 /* ===== */
78 int checkWinner() {
79     int i, line = 0;
80     for (i = 0; i < 8; i++) {
81         switch (i) {
82         case 0: line = lineSum(0, 1, 2); break;
83         case 1: line = lineSum(3, 4, 5); break;
84         case 2: line = lineSum(6, 7, 8); break;
85         case 3: line = lineSum(0, 3, 6); break;
86         case 4: line = lineSum(1, 4, 7); break;
87         case 5: line = lineSum(2, 5, 8); break;
88         case 6: line = lineSum(0, 4, 8); break;
89         case 7: line = lineSum(2, 4, 6); break;
90         }
91         if (line == 3 * MARU) return MARU;
92         else if (line == 3 * BATSU) return BATSU;
93     }
94     for (i = 0; i < 9; i++){
95         if (0 <= board[i] && board[i] < 9) return NEXT;
96     }
97     return DRAW;
98 }
99 /* ===== */
100 int main(int argc, char *argv[]){
101     /* 先後手を決定 */
102     int turn = BATSU, winner, num;
103     if (1 < argc){
104         if (!strcmp(argv[1], "-r"))
105             turn = MARU;
106     }
107     printf("スタート! [Tic Tac Toe]\n");

```

```
108     do{
109         printBoard();           /* ① 盤面の表示 */
110         num = slotNum(turn);    /* ② 手を入力 */
111         board[num] = turn;      /* ③ 手を盤面に配置 */
112         turn = switchTurn(turn); /* ④ 手番の交代 */
113         winner = checkWinner(); /* ⑤ 勝敗の判定 */
114     } while (winner == NEXT);
115     /* 対戦結果の表示 */
116     printBoard();
117     result(winner);
118     return 0;
119 }
```

A.2 スライド・パズル

ソースコード A.2 スライド・パズル

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4
5  // 定数
6  #define ROW 4
7  #define COLUMN 4
8  // 構造体
9  struct Tile{
10     int num;
11     int row;
12     int clm;
13 };
14 enum BOOLEAN{
15     false, /* false = 0, true = 1 */
16     true
17 };
18 // function prototypes
19 /*
20 int findTileNum(int);
21 void swapTile(int, int);
22 void swapTileR(int, int, int);
23 void swapTileC(int, int, int);
24 void moveTile(int);
25 void shuffle(int);
26 enum BOOLEAN check();
27 void disp();
28 void init();
29 */
30
31 static struct Tile tiles[16]; /* 16 = ROW * COLUMN */
32
33 // サブプログラム
34 int findTileNum(int num){
35     int i;
36     for (i = 0; i < ROW * COLUMN; i++){
37         if (tiles[i].num == num){
38             return i;
39         }
40     }
41     return -1;
42 }
43 /* ===== */
44 void swapTile(int n1, int n2){
45     int tmp = tiles[n1].num;
46     tiles[n1].num = tiles[n2].num;
47     tiles[n2].num = tmp;
48 }
49 /* ===== */
50 void swapTileR(int c, int r1, int r2){
51     int n1 = r1 * COLUMN + c;
52     int n2 = r2 * COLUMN + c;
53     swapTile(n1, n2);
54 }
55 /* ===== */
56 void swapTileC(int r, int c1, int c2){
57     int n1 = r * COLUMN + c1;
58     int n2 = r * COLUMN + c2;
```

```

59     swapTile(n1, n2);
60 }
61 /* ===== */
62 void moveTile(int sel){
63     int j;
64     int s = findTileNum(sel);
65     int sr = tiles[s].row;
66     int sc = tiles[s].clm;
67     int z = findTileNum(0);
68     int zr = tiles[z].row;
69     int zc = tiles[z].clm;
70     if (sr == zr){
71         if (sc < zc){
72             for (j = zc; j > sc; j--){
73                 if (0 <= j - 1){
74                     swapTileC(sr, j, j - 1);
75                 }
76             }
77         } else if (zc < sc) {
78             for (j = zc; j < sc; j++){
79                 if (j + 1 < COLUMN){
80                     swapTileC(sr, j, j + 1);
81                 }
82             }
83         }
84     }
85     if (sc == zc){
86         if (sr < zr){
87             for (j = zr; j > sr; j--){
88                 if (0 <= j - 1){
89                     swapTileR(sc, j, j - 1);
90                 }
91             }
92         } else if (zr < sr){
93             for (j = zr; j < sr; j++){
94                 if (j + 1 < ROW){
95                     swapTileR(sc, j, j + 1);
96                 }
97             }
98         }
99     }
100 }
101 /* ===== */
102 void shuffle(int n){
103     int sel, min = 0, max = ROW*COLUMN;
104     do{
105         sel = (rand() % (max - min + 1)) + min;
106         moveTile(sel);
107         n--;
108     } while (0 < n);
109 }
110 /* ===== */
111 enum BOOLEAN check(){
112     int i, j, n;
113     for (i = 0; i < ROW; i++){
114         for (j = 0; j < COLUMN; j++){
115             n = i * COLUMN + j;
116             if (tiles[n].num != n){
117                 return true;
118             }
119         }
120     }
121     return false;
122 }
123 /* ===== */

```

```
124 void disp(){
125     int i, j, n;
126     printf("\n");
127     for (i = 0; i < ROW; i++){
128         for (j = 0; j < COLUMN; j++){
129             n = i * COLUMN + j;
130             if (tiles[n].num == 0){
131                 printf("[  ]");
132             } else {
133                 printf("[%2d]", tiles[i * COLUMN + j].num);
134             }
135         }
136         printf("\n");
137     }
138 }
139 /* ===== */
140 void init(){
141     srand((unsigned)time(NULL));
142     int i, j, n;
143     for (i = 0; i < ROW; i++){
144         for (j = 0; j < COLUMN; j++){
145             n = i * COLUMN + j;
146             tiles[n].num = n;
147             tiles[n].row = i;
148             tiles[n].clm = j;
149         }
150     }
151     shuffle(1000);
152 }
153 // メイン
154 int main(void) {
155     int sel;
156     init();
157     do {
158         disp();
159         do {
160             printf("\nSelect number:");
161             scanf("%d", &sel);
162         } while( sel>ROW*COLUMN );
163         moveTile(sel);
164     } while( check() );
165
166     return 0;
167 }
```

A.3 神経衰弱

ソースコード A.3 神経衰弱

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4  #include <unistd.h>
5
6  // 定数
7  #define ROW 4
8  #define COLUMN 5
9  // 構造体
10 enum BOOLEAN {
11     false, /* false = 0, true = 1 */
12     true
13 };
14 struct Card {
15     int num;
16     int row;
17     int clm;
18     enum BOOLEAN opn;
19 };
20 // function prototypes
21 /*
22 void openCard(int);
23 void closeCard(int);
24 void switchTurn(enum BOOLEAN);
25 void disp();
26 int getNum(char*);
27 enum BOOLEAN match(int, int);
28 void wasteTime(int);
29 void swapCards(int, int);
30 void shuffle();
31 void init();
32 */
33
34 static struct Card cards[20]; /* 20 = ROW * COLUMN */
35
36 enum BOOLEAN Turn = true;
37
38 // サブプログラム
39 void openCard(int n){
40     cards[n].opn = true;
41 }
42 /* ===== */
43 void closeCard(int n){
44     cards[n].opn = false;
45 }
46 /* ===== */
47 void disp(){
48     int i, j;
49     printf("\n");
50     for (i = 0; i < ROW; i++){
51         for (j = 0; j < COLUMN; j++){
52             int n = i * COLUMN + j;
53             if ( !cards[n].opn )
54                 printf("[_]%c_", 'a' + i * COLUMN + j );
55             else
56                 printf("[_]%d_", cards[i * COLUMN + j].num );
57         }
58         printf("\n");

```

```

59     }
60     printf("\n");
61 }
62 /* ===== */
63 int getNum(char* s){
64     int cnum;
65     char str[24], work[24];
66     do {
67         sprintf(work, "%s%s%s", "\tSelect", s, "card:");
68         printf("%s", work);
69         scanf("%s", str);
70         cnum = str[0] - 'a';
71     } while (cnum > ROW * COLUMN);
72     openCard(cnum);
73     disp();
74     return cnum;
75 }
76 /* ===== */
77 enum BOOLEAN match(int n1, int n2){
78     if ( cards[n1].num == cards[n2].num )
79         return true;
80     return false;
81 }
82 /* ===== */
83 void swapCards(int n1, int n2){
84     struct Card temp;
85     temp = cards[n2];
86     cards[n2] = cards[n1];
87     cards[n1] = temp;
88 }
89 /* ===== */
90 void shuffle(){
91     int min = 0;
92     int max = ROW*COLUMN;
93     while (0 < max){
94         int sel = (rand() % (max - min + 1)) + min;
95         swapCards(sel, max--);
96     }
97 }
98 /* ===== */
99 void init(){
100     srand((unsigned)time(NULL));
101     int i, j;
102     for (i = 0; i < ROW/2; i++){
103         for (j = 0; j < COLUMN; j++){
104             int num = i * COLUMN + j;
105             cards[num].num = cards[ROW * COLUMN - num].num = num;
106             cards[num].row = cards[ROW * COLUMN - num].row = i;
107             cards[num].clm = cards[ROW * COLUMN - num].clm = j;
108             cards[num].opn = cards[ROW * COLUMN - num].opn = false;
109         }
110     }
111     shuffle();
112 }
113 // メイン
114 int main( void ){
115     int cnum1, cnum2, Closed=ROW*COLUMN;
116     int pointA=0, pointB=0;
117     init();
118     disp();
119     do{
120         printf("Aさん : %d点\tBさん : %d点\n", pointA, pointB);
121         if (Turn)
122             printf("= Aさんの番です =\n");
123         else

```



```
124     printf("= Bさんの番です \n");
125     cnum1 = getNum("1st");
126     cnum2 = getNum("2nd");
127     if ( match(cnum1, cnum2) ){
128         printf("当たり\n");
129         Closed -= 2;
130         if (Turn)
131             pointA++;
132         else
133             pointB++;
134     } else {
135         printf("ハズレ\n");
136         closeCard(cnum1);
137         closeCard(cnum2);
138         Turn = !Turn;
139     }
140     sleep(4);
141     disp();
142 } while ( 0 < Closed );
143 return 0;
144 }
```

A.4 オセロ (リバーシ)

ソースコード A.4 オセロ

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  enum BOOLEAN{
5      false,    /* false=0, true=1 */
6      true
7  };
8
9  #define BOARDW (8) // 4, 6, 8, ....
10 #define BLACK (0)
11 #define WHITE (1)
12 #define NONE (2)
13
14 int UNITV[][2] = {{0,-1},{1,-1},{1,0},{1,1},{0,1},{-1,1},{-1,0},{-1,-1}};
15 int turn = BLACK;
16 int passcount = 0;
17 int endflag = false;
18 int board[BOARDW * BOARDW];
19
20 const char* TILE[] = {
21     "●", //TILE_BLACK
22     "○", //TILE_WHITE
23     ".", //TILE_NONE
24 };
25
26 typedef struct {
27     int x, y;
28 }POS;
29
30 /* ===== */
31 void setstone(POS pos, int num){
32     int index = (pos.y * BOARDW) + pos.x;
33     board[index] = num;
34 }
35 /* ===== */
36 int getstone(POS pos){
37     int index = (pos.y * BOARDW) + pos.x;
38     return board[index];
39 }
40 /* ===== */
41 int count(int color){
42     int num=0;
43     for(int i=0; i<(BOARDW * BOARDW); i++)
44         if(board[i]==color)
45             num++;
46     return num;
47 }
48 /* ===== */
49 void drawboard(){
50     for(int x=0; x<BOARDW; x++){
51         if(x==0){
52             printf("    ");
53             char a='a';
54             for(int i=0; i<BOARDW; i++)
55                 printf("%c", a+i);
56             printf("\n");
57         }
58         printf("%2d", x+1);

```

```

59     for(int y=0; y<BOARDW; y++){
60         int index = y * BOARDW + x;
61         switch(board[index]){
62             case BLACK:printf("%s□", TILE[BLACK]);break;
63             case WHITE:printf("%s□", TILE[WHITE]);break;
64             case NONE: printf("%s□", TILE[NONE]); break;
65         }
66     }
67     printf("\n");
68 }
69 printf("\n%s(%2d), □s(%2d)", TILE[BLACK], count(BLACK), TILE[WHITE], count(WHITE));
70 if(!endflag)
71     printf("□:□turn(%s)\n", TILE[turn]);
72 else
73     printf("\n");
74 }
75 /* ===== */
76 enum BOOLEAN initboard(){
77     if((BOARDW%2)|| (BOARDW<4)|| (8<BOARDW))
78         return false;
79     int x, y;
80     POS pos;
81     for(y=0; y<BOARDW; y++){
82         for(x=0; x<BOARDW; x++){
83             pos.x = x; pos.y = y;
84             setstone(pos, NONE);
85         }
86         pos.x = pos.y = BOARDW/2-1;
87         setstone(pos, BLACK);
88         pos.x = BOARDW/2; pos.y = pos.x-1;
89         setstone(pos, WHITE);
90         pos.y = BOARDW/2; pos.x = pos.y-1;
91         setstone(pos, WHITE);
92         pos.x = pos.y = BOARDW/2;
93         setstone(pos, BLACK);
94         turn = BLACK;
95         passcount = 0;
96         endflag = false;
97         drawboard();
98         return true;
99     }
100 /* ===== */
101 POS movepos(POS pos, int v){
102     POS p;
103     p.x = pos.x + UNITV[v][0];
104     p.y = pos.y + UNITV[v][1];
105     return p;
106 }
107 /* ===== */
108 enum BOOLEAN isinside(POS pos){
109     if( (pos.x<0) || (BOARDW<=pos.x) )
110         return false;
111     if( (pos.y<0) || (BOARDW<=pos.y) )
112         return false;
113     return true;
114 }
115 /* ===== */
116 int search(POS pos, int v, int num){
117     int piece = 0;
118     while(true){
119         pos = movepos(pos, v);
120         if(!isinside(pos))
121             return 0;
122         if(getstone(pos)==NONE)
123             return 0;

```

```

124     if(getstone(pos)==num)
125         break;
126     piece ++;
127 }
128 return piece;
129 }
130 /* ===== */
131 int flippable(POS pos, int num){
132     if(getstone(pos)!=NONE)
133         return 0;
134     int total = 0;
135     int vec[]={0,0};
136     for(int i=0; i<8; i++)
137         total += search(pos, i, num);
138     return total;
139 }
140 /* ===== */
141 void nextturn(){
142     turn ^= 1;
143     int empty = 0;
144     for(int y=0; y<BOARDW; y++){
145         for(int x=0; x<BOARDW; x++){
146             POS pos; pos.x=x; pos.y=y;
147             if(getstone(pos)==NONE)
148                 empty++;
149             if(0<flippable(pos,turn)){
150                 passcount = 0;
151                 return;
152             }
153         }
154     }
155     if(empty==0){
156         endflag = true;
157         return;
158     }
159     passcount++;
160     if(2<=passcount)
161         endflag = true;
162 }
163 /* ===== */
164 POS decode(char* str){
165     POS pos;
166     pos.x = atoi(str)-1;
167     pos.y = *(str+1) - 'a';
168     return pos;
169 }
170 /* ===== */
171 void event(POS pos){
172     if(endflag){
173         initboard();
174         return;
175     }
176     if(0<passcount){
177         nextturn();
178         drawboard();
179         return;
180     }
181     if(!isinside(pos))
182         return;
183     if(0==flippable(pos, turn))
184         return;
185     for(int i=0; i<8; i++){
186         int loop = search(pos, i, turn);
187         POS temp = pos;
188         for(int j=0; j<loop; j++){
189             temp = movepos(temp, i);

```

```
189     setstone(temp, turn);
190 }
191 }
192 setstone(pos, turn);
193 nextturn();
194 drawboard();
195 }
196 /* ===== */
197 int main(void){
198     if(!initboard())
199         return 8;
200     char inpt[]="uuu";
201     while(!endflag){
202         printf("=>");
203         scanf("%s", inpt);
204         POS pos = decode(inpt);
205         printf("\n");
206         event(pos);
207     }
208     return 0;
209 }
```

参考文献

- [1] 田中賢一郎、「ゲームで学ぶ Java Script 入門」、インプレス
- [2] 松原拓也（有限会社ニコ）、「Python でリバーシを作ろう」、日経ソフトウェア 2023 年 5 月号