高知大学教育学部の情報数学のテキスト

文責　：　高知大学名誉教授　中村　治

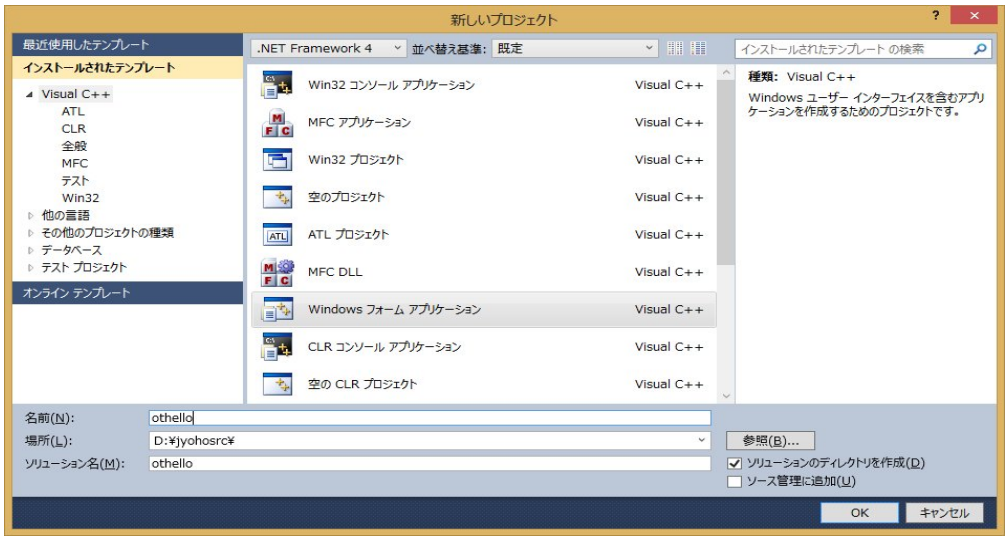
おまけ：Reversi のプログラム（ゲームでの定石の使い方と UCT の使い方のサンプル）

これはゲームのプログラミングの例ですが、長いですし、初等的なプログラミングではないです。C++ の色々な機能を使っています。UCT のプログラミングの例の私の為の備忘録だと思って下さい。興味があれば、唯、何も考えずに、そのままプログラムをコピーすれば、Reversi のプログラムが作れます。ゲームのプログラムではアルファ・ベータ法が有名で、ゲームのプログラミングの基本ですが、それは色々な本に書いてあるので、自分で読んでみたいなと思う適当な本を見付けて勉強して下さい。ここで作ったプログラムは「先手か後手か」と、「コンピュータが何も考えずに最初に見付けた手を打つか、定石とモンテカルロ法（UCT）を使って打つか」を選択できます。インターネットには、計算時間の短い多分すごく強い Reversi のプログラムがあります。このソフトを市販のオセロ（UNBALANCE のオセロ）と対戦させてみましたが、私のソフトが弱くて全然だめです。パラメータをいじれば速くなったり強くなったりしますが、本質的にこれ以上強くするにはどうすれば良いか私には分かりません。Reversi の定石や手筋は全く理解できないです。強くするには、多分、囲碁のプログラムでやっているように、オセロの戦略を強化学習で学習させ、その価値関数を UCT 探査木での事前知識として使う必要があります。私はオセロには興味がありませんからこれ以上はやりません。インターネットで得られた断片的な情報を元にこのプログラムを作りましたが、多分アルゴリズムそのものでは大きな間違いはしてないと思います。私は、将棋や囲碁はともかく、Reversi (一般にはオセロ (Othello) と呼ばれていますが、商標登録されているので、プログラムを作る人達は用心して、昔から知られているリバーシと呼んでいます) はルールしか知らないので、コンピュータが何も考えずに最初に見付けた手を打っても負けます。しかし、UCT （モンテカルロ法）を使えば、自分より強いプログラムが作れます。

今までと同様に、Microsoft Visual Studio 2010 を立ち上げ、スタートページを表示する。スタートページのタグがなければ、表示のメニューから「スタートページ」をクリックする。

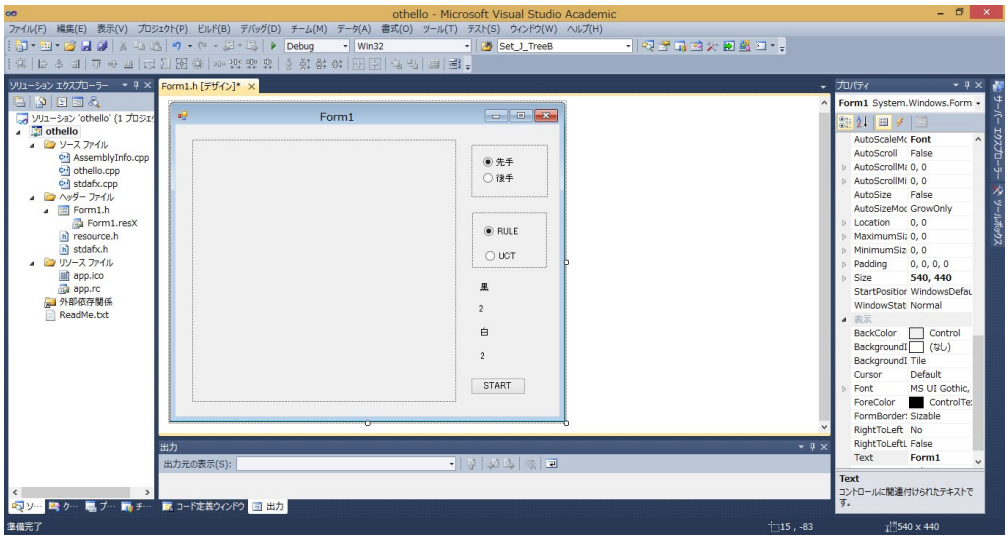
新しいプロジェクトをクリックし、

Window フォーム・アプリケーションをクリックし、名前に othello を打ち込み、

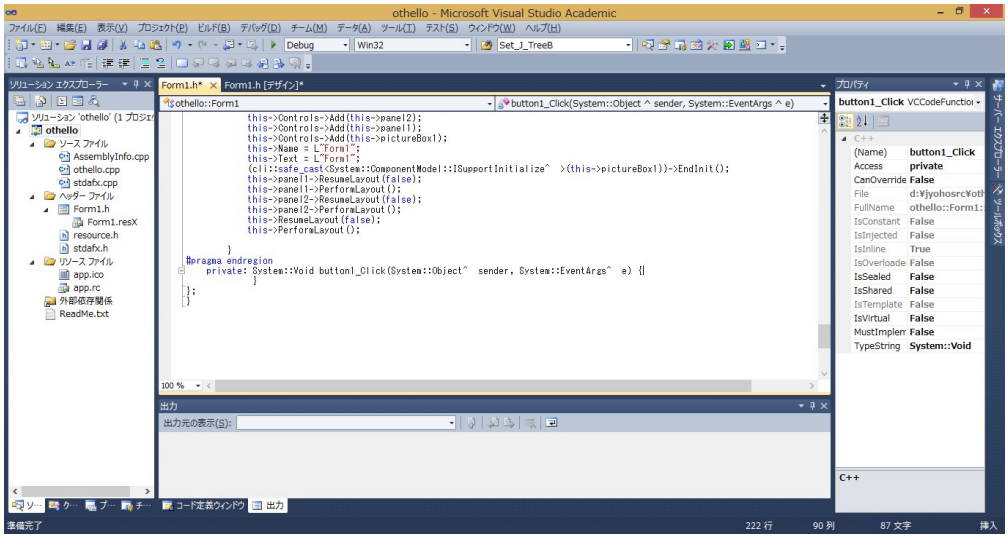


OK ボタンをクリックする。ツールボックスを表示し、Form に PictureBox 1 個と Panel の上に RadioButton 2 個と別の Panel の上に RadioButton 2 個と Label 4 個と Button 1 個とを配置する。

プロパティを表示し、 Form1 のサイズを 540, 440 とする。PictureBox のサイズを 360, 360 とする。RadioButton1 の Text を先手、Checked を True とする。RadioButton2 の Text を後手とする。。RadioButton3 の Text を RULE、Checked を True とする。RadioButton4 の Text を UCT とする。Label1 の Text を黒とし、Label2 の Text を 2 とし、Label3 の Text を白とし、Label4 の Text を 2 とする。Button1 Text を START とする。



START ボタンをダブルクリックする。

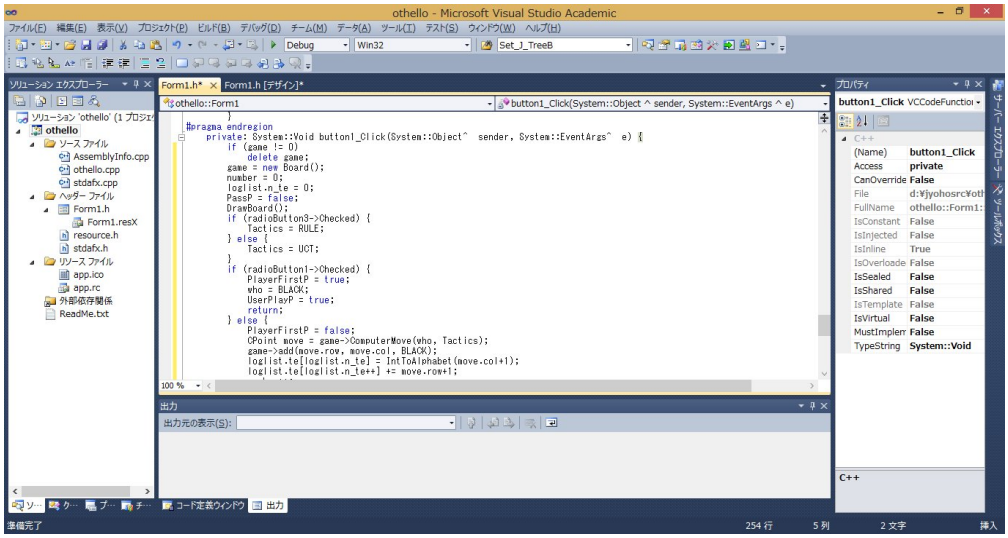



```
private: System::Void button1_Click(System::Object^  sender, System::EventArgs^  e) {
 }
```

を

```
private: System::Void button1_Click(System::Object^  sender, System::EventArgs^  e) {
    if (game != 0)
        delete game;
    game = new Board();
    number = 0;
    loglist.n_te = 0;
    PassP = false;
    DrawBoard();
    if (radioButton3->Checked) {
        Tactics = RULE;
    } else {
        Tactics = UCT;
    }
    if (radioButton1->Checked) {
        PlayerFirstP = true;
        who = BLACK;
        UserPlayP = true;
        return;
    } else {
        PlayerFirstP = false;
        CPoint move = game->ComputerMove(who, Tactics);
        game->add(move.row, move.col, BLACK);
        loglist.te[loglist.n_te] = IntToAlphabet(move.col+1);
        loglist.te[loglist.n_te++] += move.row+1;
        number++;
        DrawBoard();
        label2->Text= System::Convert::ToString(game->Calculate(BLACK));
        label4->Text= System::Convert::ToString(game->Calculate(WHITE));
        who = WHITE;
        UserPlayP = true;
        return;
    }
}
```

とする。



この

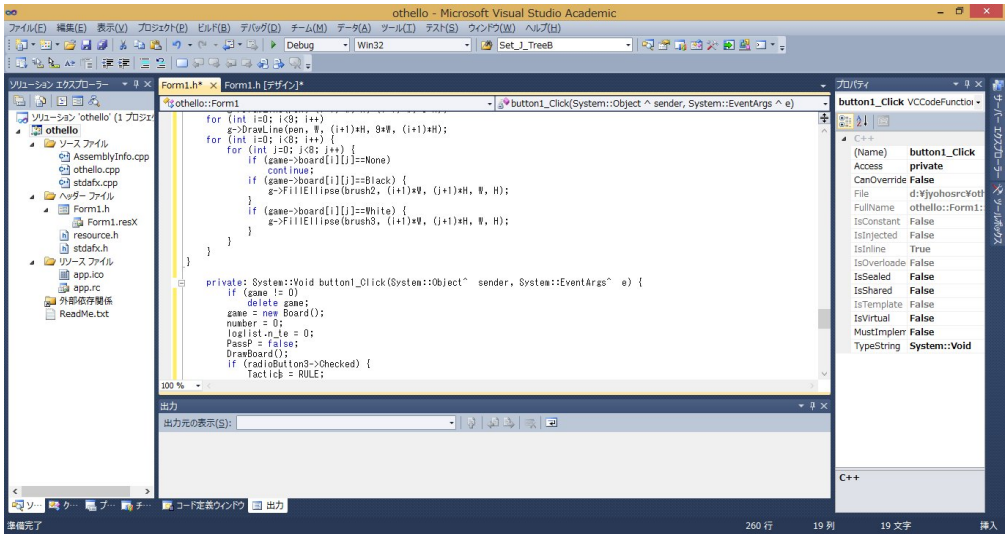
```
private: System::Void button1_Click(System::Object^  sender, System::EventArgs^  e) {
                .........
                .........
                .........
}
```

の直前に

```
void DrawBoard()
{
    Graphics^ g = pictureBox1->CreateGraphics();
    Brush^ brush  = gcnew SolidBrush(Color::Green);
    Brush^ brush2 = gcnew SolidBrush(Color::Black);
    Brush^ brush3 = gcnew SolidBrush(Color::White);
    g->FillRectangle(brush, 0, 0, pictureBox1->Width, pictureBox1->Height);
    Pen^ pen = gcnew Pen(Color::Black, 1);
    int W = pictureBox1->Width / 10;
    int H = pictureBox1->Height / 10;
    for (int i=0; i<9; i++)
        g->DrawLine(pen, (i+1)*W, H, (i+1)*W, 9*H);
    for (int i=0; i<9; i++)
        g->DrawLine(pen, W, (i+1)*H, 9*W, (i+1)*H);
    for (int i=0; i<8; i++) {
        for (int j=0; j<8; j++) {
            if (game->board[i][j]==None)
                continue;
            if (game->board[i][j]==Black) {
                g->FillEllipse(brush2, (i+1)*W, (j+1)*H, W, H);
```

```
            }
            if (game->board[i][j]==White) {
                g->FillEllipse(brush3, (i+1)*W, (j+1)*H, W, H);
            }
        }
    }
}
```

と打ち込む。



Form1.h[デザイン] のタグをクリックし、PictureBox1 をクリックしてから、プロパティのイベントのマウスの MouseDown をダブルクリックする。作られる

```
private: System::Void pictureBox1_MouseDown(System::Object^  sender,
 System::Windows::Forms::MouseEventArgs^  e) {
}
```

を

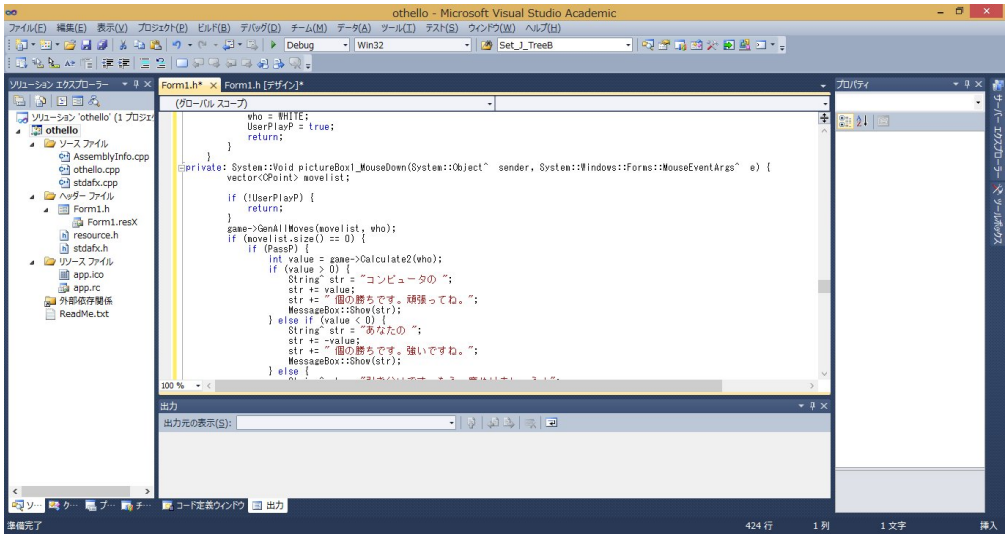
```
private: System::Void pictureBox1_MouseDown(System::Object^  sender,
 System::Windows::Forms::MouseEventArgs^  e) {
    vector<CPoint> movelist;

    if (!UserPlayP) {
        return;
    }
    game->GenAllMoves(movelist, who);
    if (movelist.size() == 0) {
        if (PassP) {
            int value = game->Calculate2(who);
            if (value > 0) {
```

```
            String^ str = "あなたの ";
            str += value;
            str += " 個の勝ちです。強いですね。";
            MessageBox::Show(str);
        } else if (value < 0) {
            String^ str = "コンピュータの ";
            str += -value;
            str += " 個の勝ちです。頑張ってね。";
            MessageBox::Show(str);
        } else {
            String^ str = "引き分けです。もう一度やりましょう！";
            MessageBox::Show(str);
        }
        return;
    } else {
        PassP = true;
        loglist.te[loglist.n_te++] = "PS";
        number++;
        if (who == BLACK) {
            who = WHITE;
        } else {
            who = BLACK;
        }
        goto LOOP;
    }
}
int W = pictureBox1->Width / 10;
int H = pictureBox1->Height / 10;
int X = e->X;
int Y = e->Y;
if ((X < W) || (X > 9*W) || (Y < H) || (Y > 9*H))
    return;
int row = X / W - 1;
int col = Y / H - 1;
if  (!game->LegalPointP(row, col, who))
    return;
PassP = false;
game->add(row, col, who);
loglist.te[loglist.n_te] = IntToAlphabet(col+1);
loglist.te[loglist.n_te++] += row+1;
number++;
DrawBoard();
label2->Text= System::Convert::ToString(game->Calculate(BLACK));
```

```
    label4->Text= System::Convert::ToString(game->Calculate(WHITE));
    if (who == BLACK) {
        who = WHITE;
    } else {
        who = BLACK;
    }

LOOP:
    UserPlayP = false;
    movelist.clear();
    game->GenAllMoves(movelist, who);
    if (movelist.size() > 0) {
        PassP = false;
        CPoint move = game->ComputerMove(who, Tactics);

        String^ str = "横=";
        str += move.row+1;
        str += " 縦=";
        str += move.col+1;
        MessageBox::Show(str);

        game->add(move.row, move.col, who);
        loglist.te[loglist.n_te] = IntToAlphabet(col+1);
        loglist.te[loglist.n_te++] += row+1;
        number++;
        DrawBoard();
        label2->Text= System::Convert::ToString(game->Calculate(BLACK));
        label4->Text= System::Convert::ToString(game->Calculate(WHITE));
        if (who == BLACK) {
            who = WHITE;
        } else {
            who = BLACK;
        }
        movelist.clear();
        game->GenAllMoves(movelist, who);
        if  (movelist.size() == 0) { // PLAYER PASS
            loglist.te[loglist.n_te++] = "PS";
            number++;
            if (who == BLACK) {
                who = WHITE;
            } else {
                who = BLACK;
            }
```

```
            goto LOOP;
        } else {
            PassP = false;
            UserPlayP = true;
            return;
        }
    }
    // COMPUTER PASS
    if (PassP) {
        int value = game->Calculate2(who);
        if (value > 0) {
            String^ str = "コンピュータの ";
            str += value;
            str += " 個の勝ちです。頑張ってね。";
            MessageBox::Show(str);
        } else if (value < 0) {
            String^ str = "あなたの ";
            str += -value;
            str += " 個の勝ちです。強いですね。";
            MessageBox::Show(str);
        } else {
            String^ str = "引き分けです。もう一度やりましょう！";
            MessageBox::Show(str);
        }
        return;
    } else {
        PassP = true;
        loglist.te[loglist.n_te++] = "PS";
        number++;
        MessageBox::Show("PASS");
        if (who == BLACK) {
            who = WHITE;
        } else {
            who = BLACK;
        }
        UserPlayP = true;
        return;
    }
}
```

とする。



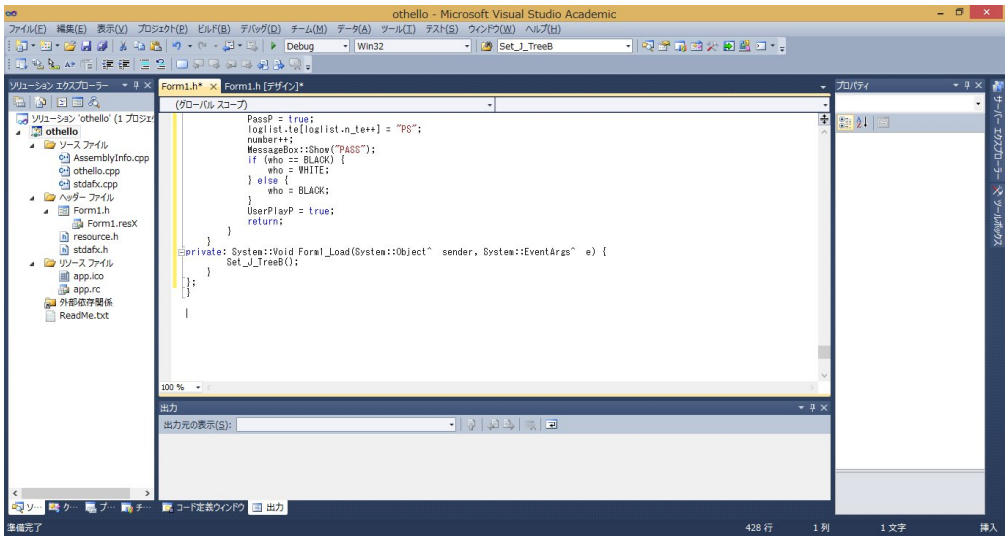
Form1.h[デザイン] のタグをクリックし、Form11 をクリックしてから、プロパティのイベントの動作の Load をダブルクリックする。作られる

```
private: System::Void Form1_Load(System::Object^  sender, System::EventArgs^  e) {
}
```

を

```
private: System::Void Form1_Load(System::Object^  sender, System::EventArgs^  e) {
    Set_J_TreeB();
}
```

とする。



最後に、めちゃくちゃ長いですが、Form1.h の最初に

```
#include <vector>
```

```
using namespace std;

typedef enum {Black, White, None} KIND;

typedef enum {BLACK, WHITE, EMPTY} WHO;

typedef enum {RULE, UCT} TACTICS;

typedef enum {NORMAL, MAIN, SUB, HALF} ROTATION;

/* Period parameters */
#define N 624
#define M 397
#define MATRIX_A 0x9908b0dfUL   /* constant vector a */
#define UPPER_MASK 0x80000000UL /* most significant w-r bits */
#define LOWER_MASK 0x7fffffffUL /* least significant r bits */

static unsigned long mt[N]; /* the array for the state vector  */
static int mti=N+1; /* mti==N+1 means mt[N] is not initialized */

/* initializes mt[N] with a seed */
void init_genrand(unsigned long s)
{
    mt[0]= s & 0xffffffffUL;
    for (mti=1; mti<N; mti++) {
        mt[mti] =
    (1812433253UL * (mt[mti-1] ^ (mt[mti-1] >> 30)) + mti);
        /* See Knuth TAOCP Vol2. 3rd Ed. P.106 for multiplier. */
        /* In the previous versions, MSBs of the seed affect   */
        /* only MSBs of the array mt[].                        */
        /* 2002/01/09 modified by Makoto Matsumoto             */
        mt[mti] &= 0xffffffffUL;
        /* for >32 bit machines */
    }
}

/* initialize by an array with array-length */
/* init_key is the array for initializing keys */
/* key_length is its length */
/* slight change for C++, 2004/2/26 */
void init_by_array(unsigned long init_key[], int key_length)
{
```

```
    int i, j, k;
    init_genrand(19650218UL);
    i=1; j=0;
    k = (N>key_length ? N : key_length);
    for (; k; k--) {
        mt[i] = (mt[i] ^ ((mt[i-1] ^ (mt[i-1] >> 30)) * 1664525UL))
          + init_key[j] + j; /* non linear */
        mt[i] &= 0xffffffffUL; /* for WORDSIZE > 32 machines */
        i++; j++;
        if (i>=N) { mt[0] = mt[N-1]; i=1; }
        if (j>=key_length) j=0;
    }
    for (k=N-1; k; k--) {
        mt[i] = (mt[i] ^ ((mt[i-1] ^ (mt[i-1] >> 30)) * 1566083941UL))
          - i; /* non linear */
        mt[i] &= 0xffffffffUL; /* for WORDSIZE > 32 machines */
        i++;
        if (i>=N) { mt[0] = mt[N-1]; i=1; }
    }

    mt[0] = 0x80000000UL; /* MSB is 1; assuring non-zero initial array */
}

/* generates a random number on [0,0xffffffff]-interval */
unsigned long genrand_int32(void)
{
    unsigned long y;
    static unsigned long mag01[2]={0x0UL, MATRIX_A};
    /* mag01[x] = x * MATRIX_A  for x=0,1 */

    if (mti >= N) { /* generate N words at one time */
        int kk;

        if (mti == N+1)   /* if init_genrand() has not been called, */
            init_genrand(5489UL); /* a default initial seed is used */

        for (kk=0;kk<N-M;kk++) {
            y = (mt[kk]&UPPER_MASK)|(mt[kk+1]&LOWER_MASK);
            mt[kk] = mt[kk+M] ^ (y >> 1) ^ mag01[y & 0x1UL];
        }
        for (;kk<N-1;kk++) {
            y = (mt[kk]&UPPER_MASK)|(mt[kk+1]&LOWER_MASK);
            mt[kk] = mt[kk+(M-N)] ^ (y >> 1) ^ mag01[y & 0x1UL];
```

```
        }
        y = (mt[N-1]&UPPER_MASK)|(mt[0]&LOWER_MASK);
        mt[N-1] = mt[M-1] ^ (y >> 1) ^ mag01[y & 0x1UL];

        mti = 0;
    }

    y = mt[mti++];

    /* Tempering */
    y ^= (y >> 11);
    y ^= (y << 7) & 0x9d2c5680UL;
    y ^= (y << 15) & 0xefc60000UL;
    y ^= (y >> 18);

    return y;
}

struct CPoint {
    int row, col;
    CPoint(){row=-1; col=-1;}
    CPoint(int r, int c){row=r; col=c;}
};

struct LOG {
    string te[100];
    int n_te;
    LOG() {n_te = 0;}
};

struct J_Tree {
    J_Tree *child;
    J_Tree *sibling;
    int row;
    int col;
    J_Tree() {row = -1; col = -1; child = 0; sibling = 0; }
    J_Tree(int x, int y) {row = y; col = x; child = 0; sibling = 0; }
};

WHO who;
int number;
bool PlayerFirstP;
bool UserPlayP;
```

```
bool PassP;
LOG loglist;
TACTICS Tactics;
ROTATION direction;

J_Tree *rootBW;
J_Tree *currentBW;

string IntToAlphabet(int n)
{
    string S;

    switch (n) {
    case 1:
        S = "a"; break;
    case 2:
        S = "b"; break;
    case 3:
        S = "c"; break;
    case 4:
        S = "d"; break;
    case 5:
        S = "e"; break;
    case 6:
        S = "f"; break;
    case 7:
        S = "g"; break;
    case 8:
        S = "h"; break;
    }
    return S;
}

void Set_J_TreeB()
{
    J_Tree *p;
    string joseki[98] = {
"f5f4e3f6d3e2f2c5f1c4e6f3c3d7",
"f5f4e3f6d3c5d6c4e6c7d7",
"f5f6e6d6c5e3d3g5e2b5",
"f5f6e6d6c5e3d3g5f3b5",
"f5f6e6d6c5e3d3g5d7c7",
"f5f6e6d6c5e3d3g5d7c6",
```

"f5f6e6d6c5e3d3c4c6b5",
"f5f6e6d6c5e3d3c4b5b6",
"f5f6e6d6c5e3e7c7d7c6",
"f5f6e6d6c5e3e7c6f7g5",
"f5f6e6d6c5e3e7f4f7f8",
"f5f6e6d6c5e3e7c4f4g6",
"f5f6e6d6c5e3e7g5f7f8",
"f5f6e6d6c5f4d7c4c3",
"f5f6e6d6c5f4e7c4d3",
"f5f6e6d6c5f4g5g4d3",
"f5f6e6d6c5f4e3c6d7",
"f5f6e6d6c5f4d3b5g4",
"f5f6e6d6c4g5c6c5b6b5a6",
"f5f6e6d6c4f4c6c5b6c3e3",
"f5f6e6d6c4g4c5f4g5e3d3",
"f5f6e6d6c6f4d7c8e8c7f7",
"f5f6e6d6c6e3f3c5e7g5g4",
"f5f6e6d6c6e3d3c5c4b5d7",
"f5f6e6d6c3f4c6d3e3d2f3",
"f5f6e6d6c3f4c6d3e3f2d7",
"f5f6e6d6c3f4c6e3d7c7g5",
"f5f6e6d6c3f4c6e3g5g4c7",
"f5f6e6d6c3g4c6f4e7d7c5",
"f5f6e6d6c3g4c6f4g5e3d3",
"f5f6e6d6c3g5c6c5c4b5b6",
"f5f6e6d6c3d3c4f3c5c6d2",
"f5f6e6d6e7g5g6e8c4f3g4",
"f5f6e6d6e7g5g6e8d7f7c6",
"f5f6e6d6e7g5g6e8h5e3d7",
"f5f6e6d6e7g5g6e8h4g4c5",
"f5f6e6d6e7g5g6e3c6d7c7",
"f5f6e6d6e7g5g6e3h5f8d8",
"f5f6e6d6e7g5g6f7f8d8",
"f5f6e6d6e7g5c5d7c4f3",
"f5f6e6d6e7g5c5f7c4e3",
"f5f6e6d6e7g5c5f7f4d3e3",
"f5f6e6d6e7g5c5f7c6f4g6",
"f5f6e6d6e7g5c5f7g6f4e8",
"f5f6e6d6e7g5c5f4c4d7",
"f5f6e6d6e7g5c5c7c6f4",
"f5f6e6d6e7f4g6h6g5f7",
"f5f6e6d6e7f4g5g6c5",
"f5f6e6d6e7f4g4g5e3",

"f5f6e6d6e7f4e3d7c4",
"f5f6e6d6e7f4c4c5e3",
"f5f6e6d6e7f4c5d7g5",
"f5f6e6d6e7f4c6f7g6",
"f5f6e6d6e7f7d7g5c5c6c7",
"f5f6e6d6e7f8c5e3",
"f5f6e6d6e7d8c5e3",
"f5f6e6d6e7f3c6f7d7f4e3",
"f5f6e6d6f7e3d7e7c6c5",
"f5f6e6d6f7f4d7g6",
"f5f6e6d6f7f4c5c6",
"f5f6e6d6f7f3e7f4e3g6g5",
"f5f6e6d6d7g5g4e7c5e3",
"f5f6e6d6c7c6f7e3f4d7",
"f5d6c3d3c4f4c5b3c2e6c6b4b5d2e3",
"f5d6c3d3c4f4c5b3c2e6c6b4b5d2a3",
"f5d6c3d3c4f4c5b3c2e6c6b4b5d2f7",
"f5d6c3d3c4f4c5b3c2e6c6b4b5d2c7",
"f5d6c3d3c4f4c5b3c2b4c6d2e6b5a5",
"f5d6c3d3c4f4c5b3c2e3d2c6b4a3g3",
"f5d6c3d3c4f4c5b4c6e6a3b5e3",
"f5d6c3d3c4f4f6g5e6c5f3b5e3",
"f5d6c3d3c4f4f6g5g6e3h5",
"f5d6c3d3c4f4f6g5c6e3d7",
"f5d6c3d3c4f4f6g5e3f3g6",
"f5d6c3d3c4f4f6g5f3g6h4",
"f5d6c3d3c4f4f6f3e6e7f7c5b6",
"f5d6c3d3c4f4f6f3e6e7d7g6e8c5c6",
"f5d6c3d3c4f4e6f6e3c5g5g3b6h6c6",
"f5d6c3d3c4f4e6b3e2e3f3c5b4a3f2",
"f5d6c3d3c4f4e3f6c6c5d7e7e6",
"f5d6c3d3c4f4f3e3c6c5f6e6b6",
"f5d6c3d3c4f4c6e3d7f6d2c5b4",
"f5d6c3d3c4f4c6c5e3f3b6b5g4",
"f5d6c3d3c4b3c6b6d7e8c2c1a3",
"f5d6c4d3c5f4e3f3c2f6d7",
"f5d6c4d3e6f4e3f3c6f6g5",
"f5d6c4g5c6c5b6c3e3b5",
"f5d6c3f4f6d3c6b6c2d2",
"f5d6c3g5c6c5c4b6f6f4",
"f5d6c5f4e3c6d3f6e6d7g3c4g5",
"f5d6c5f4e3c6d3f6e6d7e7c7c4",
"f5d6c5f4e3c6d3f3e6f7g4c3c2",

```
"f5d6c5f4e3c6d3g5f6f3g6c4c3",
"f5d6c5f4e3c6e6f7d7e8f3f6",
"f5d6c5f4e3c6e6f6g3c4",
"f5d6c5f4e3d3f3e2c4c3f2b3e1",
"f5d6c5f4e3d3e6g5c4c3d2f6c2",
"f5d6c5f6e6f4c6e7f8c4c3"
    };

    rootBW = 0;
    for(int i=0; i<98; i++) {
        string str = joseki[i];
        int len = str.length();
        p = rootBW;
        for (int i=0; i<len/2; i++) {
            int col = str[2*i]-'a';
            int row = str[2*i+1]-'1';
            if (p == 0) {
                rootBW = new J_Tree(col, row);
                p = rootBW;
                continue;
            } else {
                if (p->col == col && p->row == row) {
                    continue;
                }
                if (p->child == 0) {
                    p->child = new J_Tree(col, row);
                    p = p->child;
                    continue;
                } else if (p->child->col == col && p->child->row == row) {
                    p = p->child;
                    continue;
                } else {
                    bool flag = false;
                    p = p->child;
                    while (p->sibling != 0) {
                        if (p->sibling->col == col && p->sibling->row == row) {
                            p = p->sibling;
                            flag = true;
                            break;
                        }
                        p = p->sibling;
                    }
                    if (!flag) {
```

```
                        p->sibling = new J_Tree(col, row);
                        p = p->sibling;
                    }
                }
            }
        }
    }
}

#define Max  99999
#define Min -99999
#define uct_max 100
#define max_depth 12
#define MAXUCTLOOP 10000

bool OnePlayOutPassP;

struct UCT_Tree {
    double UCB;
    double X;
    int row;
    int col;
    WHO who;
    int n;
    int CN;
    UCT_Tree * parent;
    UCT_Tree * child;
    UCT_Tree * next;
    UCT_Tree(){row=-1; col=-1; n=0; CN=0; parent=0; child=0; next=0;}
};

struct Board {
    KIND board[8][8];
    UCT_Tree *tree;
    Board();
    void add(int row, int col, WHO who);
    int Calculate(WHO who);
    int Calculate2(WHO who);
    bool LegalPointP(int row, int col, WHO who);
    void GenAllMoves(vector<CPoint> &movelist, WHO who);
    CPoint ComputerMove(WHO who, TACTICS tactics);
    void delete_UCT_Tree(UCT_Tree *tree);
    int OnePlayOut(CPoint pt, WHO who, WHO origin);
```

```
    double change_UCT_Tree(UCT_Tree *tree, bool PositiveP);
};

Board::Board()
{
    for (int i=0; i<8; i++)
        for (int j=0; j<8; j++)
            board[i][j] = None;
    board[3][3] = White;
    board[4][3] = Black;
    board[3][4] = Black;
    board[4][4] = White;
    tree = 0;
}

void Board::add(int row, int col, WHO player)
{
    KIND playerPoint, enemyPoint;
    bool flag;
    int k;

    if (player == BLACK) {
        playerPoint = Black;
        enemyPoint = White;
    } else {
        playerPoint = White;
        enemyPoint = Black;
    }

    board[row][col] = playerPoint;

    if  ((col-1>=0) && (board[row][col-1]==enemyPoint)) {
        flag = false;
        for (int i=2; col-i >= 0; i++) {
            if (board[row][col-i]==playerPoint) {
                flag = true;
                break;
            } else if (board[row][col-i]==None)
                break;
        }
        if (flag) {
            k = 1;
            while (board[row][col-k]==enemyPoint) {
```

```
                board[row][col-k] = playerPoint;
                k++;
            }
        }
    }
    if ((col+1<8) && (board[row][col+1]==enemyPoint)) {
        flag = false;
        for (int i=2; col+i < 8; i++) {
            if (board[row][col+i]==playerPoint) {
                flag = true;
                break;
            } else if (board[row][col+i]==None)
                break;
        }
        if (flag) {
            k = 1;
            while (board[row][col+k]==enemyPoint) {
                board[row][col+k] = playerPoint;
                k++;
            }
        }
    }
    if ((row-1>=0) && (board[row-1][col]==enemyPoint)) {
        flag = false;
        for (int i=2; row-i >= 0; i++) {
            if (board[row-i][col]==playerPoint) {
                flag = true;
                break;
            } else if (board[row-i][col]==None)
                break;
        }
        if (flag) {
            k = 1;
            while (board[row-k][col]==enemyPoint) {
                board[row-k][col] = playerPoint;
                k++;
             }
        }
    }
    if ((row+1<8) && (board[row+1][col]==enemyPoint)) {
        flag = false;
        for (int i=2; row+i < 8; i++) {
            if (board[row+i][col]==playerPoint) {
```

```
            flag = true;
            break;
        } else if (board[row+i][col]==None)
            break;
    }
    if (flag) {
        k = 1;
        while (board[row+k][col]==enemyPoint) {
            board[row+k][col] = playerPoint;
            k++;
         }
    }
}
if (((row-1>=0)&&(col-1>=0)) && (board[row-1][col-1]==enemyPoint)) {
    flag = false;
    for (int i=2; (row-i>=0)&&(col-i>=0); i++) {
        if (board[row-i][col-i]==playerPoint) {
            flag = true;
            break;
        } else if (board[row-i][col-i]==None)
            break;
    }
    if (flag) {
        k = 1;
        while (board[row-k][col-k]==enemyPoint) {
            board[row-k][col-k] = playerPoint;
            k++;
        }
    }
}
if (((row-1>=0)&&(col+1<8)) && (board[row-1][col+1]==enemyPoint)) {
    flag = false;
    for (int i=2; (row-i>=0)&&(col+i<8); i++) {
        if (board[row-i][col+i]==playerPoint) {
            flag = true;
            break;
        } else if (board[row-i][col+i]==None)
            break;
    }
    if (flag) {
        k = 1;
        while (board[row-k][col+k]==enemyPoint) {
            board[row-k][col+k] = playerPoint;
```

```
            k++;
        }
    }
}
if (((row+1<8)&&(col-1>=0)) && (board[row+1][col-1]==enemyPoint)) {
    flag = false;
    for (int i=2; (row+i<8)&&(col-i>=0); i++) {
        if (board[row+i][col-i]==playerPoint) {
            flag = true;
            break;
        } else if (board[row+i][col-i]==None)
            break;
        }
    if (flag) {
        k = 1;
        while (board[row+k][col-k]==enemyPoint) {
            board[row+k][col-k] = playerPoint;
            k++;
        }
    }
}
if (((row+1<8)&&(col+1<8)) && (board[row+1][col+1]==enemyPoint)) {
    flag = false;
    for (int i=2; (row+i<8)&&(col+i<8); i++) {
        if (board[row+i][col+i]==playerPoint) {
            flag = true;
            break;
        } else if (board[row+i][col+i]==None)
            break;
    }
    if (flag) {
        k = 1;
        while (board[row+k][col+k]==enemyPoint) {
            board[row+k][col+k] = playerPoint;
            k++;
         }
    }
}
if (((row-1>=0)&&(col+1<8)) && (board[row-1][col+1]==enemyPoint)) {
    flag = false;
    for (int i=2; (row-i>=0)&&(col+i<8); i++) {
        if (board[row-i][col+i]==playerPoint) {
            flag = true;
```

```
                break;
            } else if (board[row-i][col+i]==None)
                break;
        }
        if (flag) {
            k = 1;
            while (board[row-k][col+k]==enemyPoint) {
                board[row-k][col+k] = playerPoint;
                k++;
            }
        }
    }
}

int Board::Calculate(WHO who)
{
    KIND playerPoint, enemyPoint;

    if (who == BLACK) {
        playerPoint = Black;
        enemyPoint = White;
    } else {
        playerPoint = White;
        enemyPoint = Black;
    }
    int Pnum=0;
    for (int row=0; row<8; row++)
        for (int col=0; col<8; col++) {
            if (board[row][col] == playerPoint)
                Pnum ++;
        }
    return Pnum;
}

int Board::Calculate2(WHO who)
{
    KIND playerPoint, enemyPoint;

    if (who == BLACK) {
        playerPoint = Black;
        enemyPoint = White;
    } else {
        playerPoint = White;
```

```
        enemyPoint = Black;
    }
    int Pnum=0, Enum=0;
    for (int row=0; row<8; row++)
        for (int col=0; col<8; col++) {
            if (board[row][col] == playerPoint)
                Pnum++;
            else if (board[row][col] == enemyPoint)
                Enum++;
        }
    return Pnum-Enum;
}

bool Board::LegalPointP(int row, int col, WHO player)
{
    bool flag;
    KIND playerPoint, enemyPoint;

    if (board[row][col] != None)
        return false;
    if (player == BLACK) {
        playerPoint = Black;
        enemyPoint = White;
    } else {
        playerPoint = White;
        enemyPoint = Black;
    }
    if ((col-1>=0) && (board[row][col-1]==enemyPoint)) {
        flag = false;
        for (int i=2; col-i >= 0; i++) {
            if (board[row][col-i]==playerPoint) {
                flag = true;
                break;
            } else if (board[row][col-i]==None)
                break;
        }
        if (flag)
            return true;
    }
    if ((col+1<8) && (board[row][col+1]==enemyPoint)) {
        flag = false;
        for (int i=2; col+i < 8; i++) {
            if (board[row][col+i]==playerPoint) {
```

```
            flag = true;
            break;
        } else if (board[row][col+i]==None)
            break;
    }
    if (flag)
        return true;
}
if ((row-1>=0) && (board[row-1][col]==enemyPoint)) {
    flag = false;
    for (int i=2; row-i >= 0; i++) {
        if (board[row-i][col]==playerPoint) {
            flag = true;
            break;
        } else if (board[row-i][col]==None)
            break;
    }
    if (flag) {
        return true;
    }
}
if ((row+1<8) && (board[row+1][col]==enemyPoint)) {
    flag = false;
    for (int i=2; row+i < 8; i++) {
        if (board[row+i][col]==playerPoint) {
            flag = true;
            break;
        } else if (board[row+i][col]==None)
            break;
    }
    if (flag)
        return true;
}
if (((row-1>=0)&&(col-1>=0)) && (board[row-1][col-1]==enemyPoint)) {
    flag = false;
    for (int i=2; (row-i>=0)&&(col-i>=0); i++) {
        if (board[row-i][col-i]==playerPoint) {
            flag = true;
            break;
        } else if (board[row-i][col-i]==None)
            break;
    }
    if (flag) {
```

```
            return true;
        }
    }
    if (((row-1>=0)&&(col+1<8)) && (board[row-1][col+1]==enemyPoint)) {
        flag = false;
        for (int i=2; (row-i>=0)&&(col+i<8); i++) {
            if (board[row-i][col+i]==playerPoint) {
                flag = true;
                break;
            } else if (board[row-i][col+i]==None)
                break;
        }
        if (flag)
            return true;
    }
    if (((row+1<8)&&(col-1>=0)) && (board[row+1][col-1]==enemyPoint)) {
        flag = false;
        for (int i=2; (row+i<8)&&(col-i>=0); i++) {
            if (board[row+i][col-i]==playerPoint) {
                flag = true;
                break;
            } else if (board[row+i][col-i]==None)
                break;
        }
        if (flag)
            return true;
    }
    if (((row+1<8)&&(col+1<8)) && (board[row+1][col+1]==enemyPoint)) {
        flag = false;
        for (int i=2; (row+i<8)&&(col+i<8); i++) {
            if (board[row+i][col+i]==playerPoint) {
                flag = true;
                break;
            } else if (board[row+i][col+i]==None)
                break;
        }
        if (flag)
            return true;
    }
    if (((row-1>=0)&&(col+1<8)) && (board[row-1][col+1]==enemyPoint)) {
        flag = false;
        for (int i=2; (row-i>=0)&&(col+i<8); i++) {
            if (board[row-i][col+i]==playerPoint) {
```

```
                flag = true;
                break;
            } else if (board[row-i][col+i]==None)
                break;
        }
        if (flag)
            return true;
    }
    return false;
}

void Board::GenAllMoves(vector<CPoint> &movelist, WHO who)
{
    for (int row=0; row<8; row++)
        for (int col=0; col<8; col++)
            if (LegalPointP(row, col, who))
                movelist.push_back(CPoint(row, col));
}

void Board::delete_UCT_Tree(UCT_Tree *tree)
{
    UCT_Tree *p;
    while (tree != 0) {
        p = tree;
        tree = tree->next;
        if (p->child != 0)
            delete_UCT_Tree(p->child);
        delete p;
    }
}

int Board::OnePlayOut(CPoint pt, WHO who, WHO origin)
{
    vector<CPoint> movelist;
    CPoint move;
    int result;

    add(pt.row, pt.col, who);
    OnePlayOutPassP = false;

GG: if (who == BLACK) {
        who = WHITE;
    } else {
```

```
        who = BLACK;
    }
    movelist.clear();
    GenAllMoves(movelist, who);
    if (movelist.size() == 0) {
        if (OnePlayOutPassP) {
            result = Calculate2(origin);
            if (result > 0)
                result = 1;
            else if (result < 0)
                result = -1;
            return result;
        } else {
            OnePlayOutPassP = true;
            goto GG;
        }
    }

    int ind = genrand_int32() % movelist.size();

    move = movelist[ind];

    result = OnePlayOut(move, who, origin);
    return result;
}

CPoint Board::ComputerMove(WHO who, TACTICS tactics)
{
    vector<CPoint> movelist;
    CPoint move;
    char s[10];
    int col, row;

    if (tactics == RULE) {
        GenAllMoves(movelist, who);
        if (movelist.size() == 0)
            return move;
        return movelist[0];
    } else if (tactics == UCT) {
        if (who == BLACK && number == 0) {
            move.row = 4;
            move.col = 5;  // f5
            currentBW = rootBW;
```

```
    direction = NORMAL;
    return move;
}
if (who == WHITE && number == 1) {
    currentBW = rootBW;
    strcpy(s, loglist.te[loglist.n_te-1].c_str());
    col = s[0] - 'a';
    row = s[1] - 1;
    if (col == 5 && row == 4) {
        direction = NORMAL;
    } else if (col == 4 && row == 5) {
        direction = MAIN;
    } else if (col == 3 && row == 2) {
        direction = SUB;
    } else if (col == 2 && row == 3) {
        direction = HALF;
    }
}

strcpy(s, loglist.te[loglist.n_te-1].c_str());
col = s[0] - 'a';
row = s[1] - 1;
if (direction == MAIN) {
    int junk = col;
    col = row;
    row = junk;
} else if (direction == SUB) {
    int junk = col;
    col = 7 - row;
    row = 7 - junk;
} else if (direction == HALF) {
    col = 7 - col;
    row = 7 - row;
}

if (currentBW != 0 && currentBW->child != 0) { // 定石手順
    if (currentBW->col == col && currentBW->row == row) {
        currentBW = currentBW->child;
        if (currentBW != 0) {
            int count = 1;
            J_Tree *p = currentBW;
            while (p->sibling != 0) {
                count++;
```

```
            p = p->sibling;
        }
        int index = genrand_int32() % count;
        index++;
        if (index == 1) {
            if (direction == NORMAL) {
                move.row = currentBW->row;
                move.col = currentBW->col;
            } else if (direction == MAIN) {
                move.row = currentBW->col;
                move.col = currentBW->row;
            } else if (direction == SUB) {
                move.row = 7-currentBW->col;
                move.col = 7-currentBW->row;
            } else if (direction == HALF) {
                move.row = 7-currentBW->row;
                move.col = 7-currentBW->col;
            }
            return move;
        } else {
            p = currentBW;
            int cnt = 1;
            while (cnt < index) {
                p = p->sibling;
                cnt++;
            }
            currentBW = p;
            if (direction == NORMAL) {
                move.row = currentBW->row;
                move.col = currentBW->col;
            } else if (direction == MAIN) {
                move.row = currentBW->col;
                move.col = currentBW->row;
            } else if (direction == SUB) {
                move.row = 7-currentBW->col;
                move.col = 7-currentBW->row;
            } else if (direction == HALF) {
                move.row = 7-currentBW->row;
                move.col = 7-currentBW->col;
            }
            return move;
        }
    } else {
```

```
            currentBW = 0;
        }
    } else if (currentBW->child->col == col && currentBW->child->row == row) {
        currentBW = currentBW->child;
        if (currentBW->child != 0) {
            currentBW = currentBW->child;
            int count = 1;
            J_Tree *p = currentBW;
            while (p->sibling != 0) {
                count++;
                p = p->sibling;
             }
             int index = genrand_int32() % count;
             index++;
             if (index == 1) {
                 if (direction == NORMAL) {
                     move.row = currentBW->row;
                     move.col = currentBW->col;
                 } else if (direction == MAIN) {
                     move.row = currentBW->col;
                     move.col = currentBW->row;
                 } else if (direction == SUB) {
                     move.row = 7-currentBW->col;
                     move.col = 7-currentBW->row;
                 } else if (direction == HALF) {
                     move.row = 7-currentBW->row;
                     move.col = 7-currentBW->col;
                 }
                 return move;
            } else {
                 p = currentBW;
                 int cnt=1;
                 while (cnt < index) {
                     p = p->sibling;
                     cnt++;
                  }
                  currentBW = p;
                  if (direction == NORMAL) {
                      move.row = currentBW->row;
                      move.col = currentBW->col;
                  } else if (direction == MAIN) {
                      move.row = currentBW->col;
                      move.col = currentBW->row;
```

```
                } else if (direction == SUB) {
                    move.row = 7-currentBW->col;
                    move.col = 7-currentBW->row;
                } else if (direction == HALF) {
                    move.row = 7-currentBW->row;
                    move.col = 7-currentBW->col;
                }
                return move;
            }
        } else {
            currentBW = 0;
        }
    } else {
        currentBW = currentBW->child->sibling;
        while (currentBW != 0) {
            if (currentBW->col == col && currentBW->row == row) {
                if (currentBW->child != 0) {
                    currentBW = currentBW->child;
                    int count = 1;
                    J_Tree *p = currentBW;
                    while (p->sibling != 0) {
                        count++;
                        p = p->sibling;
                    }
                    int index = genrand_int32() % count;
                    index++;
                    if (index == 1) {
                        if (direction == NORMAL) {
                            move.row = currentBW->row;
                            move.col = currentBW->col;
                        } else if (direction == MAIN) {
                            move.row = currentBW->col;
                            move.col = currentBW->row;
                        } else if (direction == SUB) {
                            move.row = 7-currentBW->col;
                            move.col = 7-currentBW->row;
                        } else if (direction == HALF) {
                            move.row = 7-currentBW->row;
                            move.col = 7-currentBW->col;
                        }
                        return move;
                    } else {
                        p = currentBW;
```

```
                        int cnt = 1;
                        while (cnt < index) {
                            p = p->sibling;
                            cnt++;
                        }
                        currentBW = p;
                        if (direction == NORMAL) {
                            move.row = currentBW->row;
                            move.col = currentBW->col;
                        } else if (direction == MAIN) {
                            move.row = currentBW->col;
                            move.col = currentBW->row;
                        } else if (direction == SUB) {
                            move.row = 7-currentBW->col;
                            move.col = 7-currentBW->row;
                        } else if (direction == HALF) {
                            move.row = 7-currentBW->row;
                            move.col = 7-currentBW->col;
                        }
                        return move;
                    }
                } else {
                    currentBW = 0;
                    break;
                }
            }
            currentBW = currentBW->sibling;
        }
    }
}

UCT_Tree *pp, *maxpp;
WHO currentWho = who;

if (tree != 0) {
    delete_UCT_Tree(tree);
}

tree = 0;
movelist.clear();
GenAllMoves(movelist, who);
if (movelist.size() == 0)
    return move;
```

```
        if (movelist.size() == 1) {
            return movelist[0];
        }
        for (int i=0; i<movelist.size(); i++) {
            pp = new UCT_Tree();
            pp->row = movelist[i].row;
            pp->col = movelist[i].col;
            pp->who = who;
            pp->UCB = 500.0 + genrand_int32() % 20;
            pp->X = 0.0;
            pp->n = 0;
            pp->CN = 0;
            pp->parent = 0;
            pp->child = 0;
            pp->next = tree;
            tree = pp;
        }
        double MaxUCB;
        int depth;

        for (int k=0; k<MAXUCTLOOP; k++) {
            Board oldboard;
            for (int i=0; i<8; i++)
                for (int j=0; j<8; j++)
                    oldboard.board[i][j] = board[i][j];
            pp = tree;
            depth = 0;
LOOPA:      MaxUCB = pp->UCB;
            pp->CN++;
            maxpp = pp;
            depth++;
            pp = pp->next;
            while (pp != 0) {
                if (currentWho == pp->who) {
                    if (pp->UCB > MaxUCB) {
                        MaxUCB = pp->UCB;
                        maxpp = pp;
                    }
                } else {
                    if (pp->UCB < MaxUCB) {
                        MaxUCB = pp->UCB;
                        maxpp = pp;
                     }
```

```
    }
    pp->CN++;
    pp = pp->next;
}
if (maxpp->child != 0) {
    add(maxpp->row, maxpp->col, maxpp->who);
    maxpp->n++;
    pp = maxpp->child;
    goto LOOPA;
 } else { // maxpp->child == 0
    maxpp->n++;
    if (depth < max_depth && maxpp->n > uct_max) {
        // 新しい子供を作る
        if (maxpp->who == BLACK)
            who = WHITE;
        else
            who = BLACK;
        add(maxpp->row, maxpp->col, maxpp->who);
        movelist.clear();
        GenAllMoves(movelist, who);
        if (movelist.size() == 0) { // leaf or PASS
            // leaf?
            int count = 0;
            for (int i=0; i<8; i++) {
                for (int j=0; j<8; j++) {
                    if (board[i][j] == None) {
                        count++;
                    }
                 }
            }
            if (count > 0) {
                goto HHH;
            }
            int result = Calculate2(currentWho);
            if (result > 0)
                result = 1;
            else if (result < 0)
                result = -1;
            // UCB の更新 と pp->X の修正
            maxpp->X = (maxpp->X*(maxpp->n-1)+result)/(maxpp->n);
            if (maxpp->parent == 0) {
                pp = tree;
            } else {
```

```
                pp = maxpp->parent;
                pp = pp->child;
            }
            while (pp != 0) {
                if (pp->n > 0) {
                    if (currentWho == pp->who) {
                        pp->UCB = pp->X+sqrt(2.0*log((double)pp->CN)/(pp->n));
                    } else {
                        pp->UCB = pp->X-sqrt(2.0*log((double)pp->CN)/(pp->n));
                    }
                }
                pp = pp->next;
            }
            // 親の pp->UCB と pp->X の修正
            UCT_Tree *qq = maxpp->parent;
            while (qq != 0) {
                qq->X = (qq->X*(qq->n-1)+result)/(qq->n);
                if (qq->parent == 0) {
                    pp = tree;
                } else {
                    pp = qq->parent;
                    pp = pp->child;
                }
                while (pp != 0) {
                    if (currentWho == pp->who) {
                        if (pp->n > 0)
                            pp->UCB = pp->X+sqrt(2.0*log((double)pp->CN)/(pp->n));
                     } else {
                        if (pp->n > 0)
                            pp->UCB = pp->X-sqrt(2.0*log((double)pp->CN)/(pp->n));
                     }
                     pp = pp->next;
                }
                qq = qq->parent;
            }
            for (int i=0; i<8; i++)
                for (int j=0; j<8; j++)
                    board[i][j] = oldboard.board[i][j];
            continue;
        } else { // non leaf
            for (int i=0; i<movelist.size(); i++) {
                pp = new UCT_Tree();
                pp->row = movelist[i].row;
```

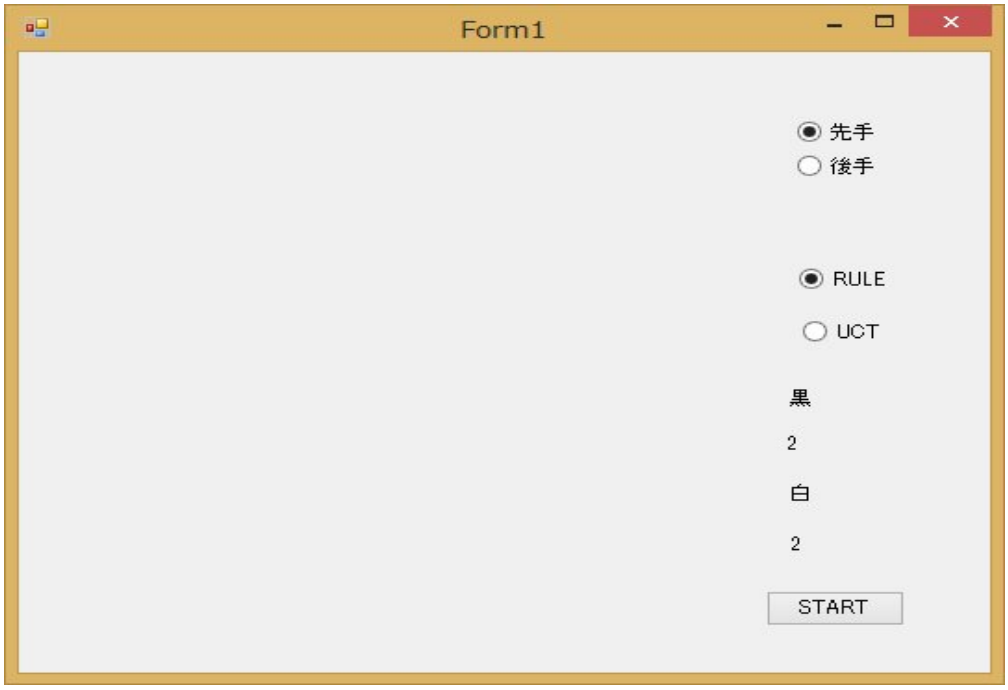
```
    pp->col = movelist[i].col;
    pp->who = (maxpp->who==BLACK)?WHITE:BLACK;
    if (currentWho == pp->who) {
        pp->UCB = 500.0 + genrand_int32() % 20;
    } else {
        pp->UCB = -500.0 + genrand_int32() % 20;
    }
    pp->X = 0.0;
    pp->n = 0;
    pp->CN = 1;
    pp->parent = maxpp;
    pp->child = 0;
    pp->next = maxpp->child;
    maxpp->child = pp;
}
pp = maxpp->child;
MaxUCB = pp->UCB;
maxpp = pp;
pp = pp->next;
while (pp != 0) {
    if (currentWho == pp->who) {
        if (pp->UCB > MaxUCB) {
            MaxUCB = pp->UCB;
            maxpp = pp;
        }
     } else {
        if (pp->UCB < MaxUCB) {
            MaxUCB = pp->UCB;
            maxpp = pp;
        }
    }
    pp = pp->next;
}
CPoint C;
C.row = maxpp->row;
C.col = maxpp->col;
OnePlayOutPassP = false;
int result = OnePlayOut(C, maxpp->who, currentWho);
// UCB と pp->X の修正
maxpp->n++;
maxpp->X = (maxpp->X*(maxpp->n-1)+result)/(maxpp->n);
if (maxpp->parent == 0) {
    pp = tree;
```

```
            } else {
                pp = maxpp->parent;
                pp = pp->child;
            }
            while (pp != 0) {
                if (pp->n > 0) {
                    if (currentWho == pp->who) {
                        pp->UCB = pp->X+sqrt(2.0*log((double)pp->CN)/(pp->n));
                    } else {
                        pp->UCB = pp->X-sqrt(2.0*log((double)pp->CN)/(pp->n));
                    }
                }
                pp = pp->next;
            }
            // 親の pp->UCB と pp->X の修正
            UCT_Tree *qq = maxpp->parent;
            while (qq != 0) {
                qq->X = (qq->X*(qq->n-1)+result)/(qq->n);
                if (qq->parent == 0) {
                    pp = tree;
                } else {
                    pp = qq->parent;
                    pp = pp->child;
                }
                while (pp != 0) {
                    if (currentWho == pp->who) {
                        if (pp->n > 0)
                            pp->UCB = pp->X+sqrt(2.0*log((double)pp->CN)/(pp->n));
                    } else {
                        if (pp->n > 0)
                            pp->UCB = pp->X-sqrt(2.0*log((double)pp->CN)/(pp->n));
                    }
                    pp = pp->next;
                 }
                 qq = qq->parent;
            }
            for (int i=0; i<8; i++)
                for (int j=0; j<8; j++)
                    board[i][j] = oldboard.board[i][j];
            continue;
        }
    } else {
        // 新しい子供は作らない  maxpp->child == 0
```

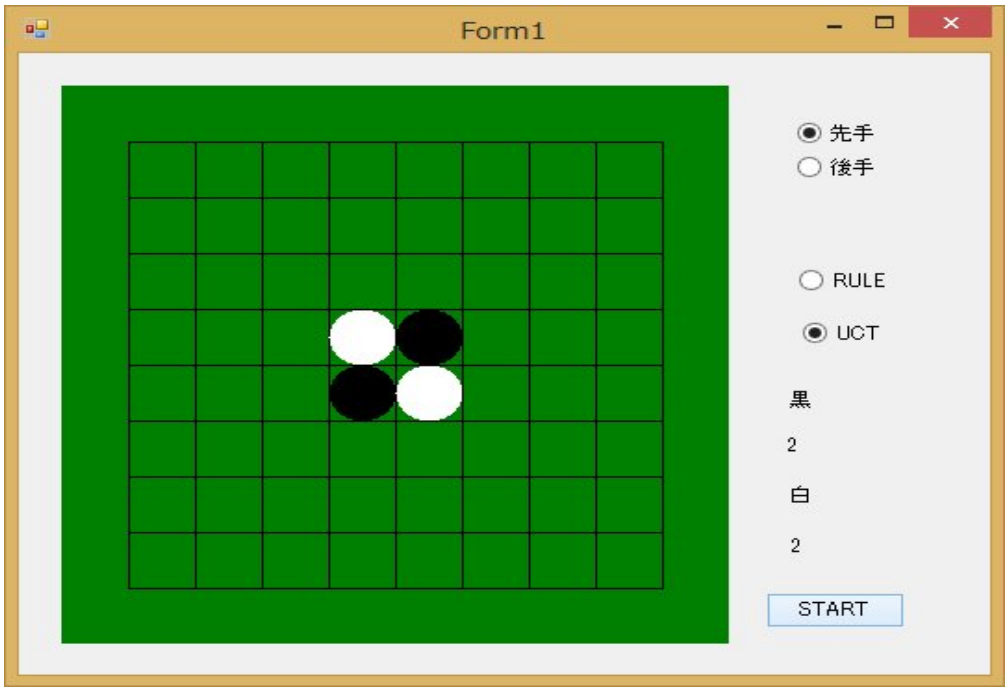
```
                    // maxpp->N  && maxpp->n は増やしている
HHH:                CPoint C;
                    C.row = maxpp->row;
                    C.col = maxpp->col;
                    OnePlayOutPassP = false;
                    int result = OnePlayOut(C, maxpp->who, currentWho);
                    // UCB と pp->X の修正
                    maxpp->X = (maxpp->X*(maxpp->n-1)+result)/(maxpp->n);
                    if (maxpp->parent == 0) {
                        pp = tree;
                    } else {
                        pp = maxpp->parent;
                        pp = pp->child;
                    }
                    while (pp != 0) {
                        if (pp->n > 0) {
                            if (currentWho == pp->who) {
                                pp->UCB = pp->X+sqrt(2.0*log((double)pp->CN)/(pp->n));
                            } else {
                                pp->UCB = pp->X-sqrt(2.0*log((double)pp->CN)/(pp->n));
                            }
                        }
                        pp = pp->next;
                    }
                    // 親の pp->UCB と pp->X の修正
                    UCT_Tree *qq = maxpp->parent;
                    while (qq != 0) {
                        qq->X = (qq->X*(qq->n-1)+result)/(qq->n);
                        if (qq->parent == 0) {
                            pp = tree;
                        } else {
                            pp = qq->parent;
                            pp = pp->child;
                        }
                        while (pp != 0) {
                            if (currentWho == pp->who) {
                                if (pp->n > 0)
                                    pp->UCB = pp->X+sqrt(2.0*log((double)pp->CN)/(pp->n));
                            } else {
                                if (pp->n > 0)
                                    pp->UCB = pp->X-sqrt(2.0*log((double)pp->CN)/(pp->n));
                            }
                            pp = pp->next;
```

```
                    }
                    qq = qq->parent;
                }
                for (int i=0; i<8; i++)
                    for (int j=0; j<8; j++)
                        board[i][j] = oldboard.board[i][j];
                continue;
            }
        }
    }

    pp = tree;
    int maxN = pp->n;
    maxpp = pp;
    pp = pp->next;
    while (pp != 0) {
        if (maxN < pp->n) {
            maxN = pp->n;
            maxpp = pp;
        }
        pp = pp->next;
    }
    CPoint C;
    C.row = maxpp->row;
    C.col = maxpp->col;
    move = C;

    delete_UCT_Tree(tree);
    tree = 0;
    return move;
  }
}

Board *game;
```

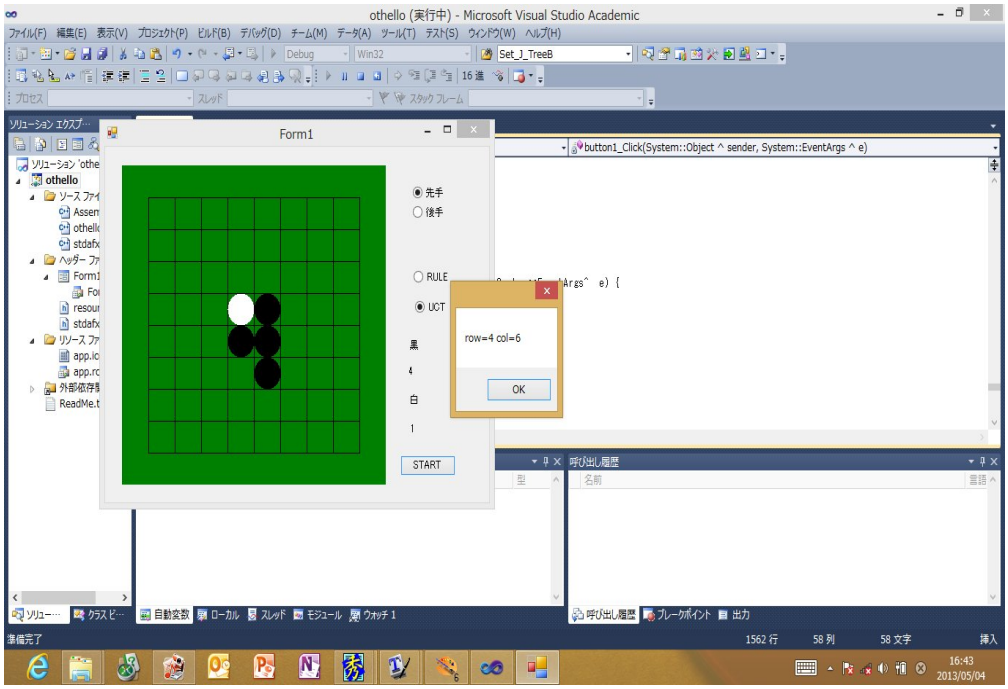
と打ち込む。
　実行する。



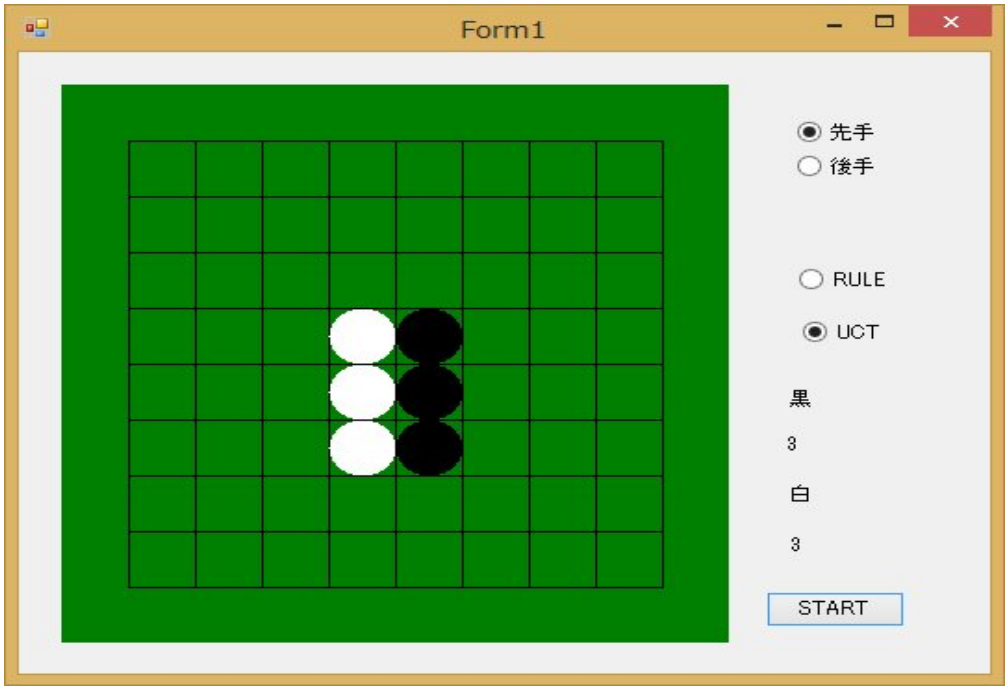
自分が先手か後手かとコンピュータが出鱈目に打つかモンテカルロ法で打つかを決めて、START ボタンをクリックする。



打ちたい所をクリックする。コンピュータが手を決めたらメセージボックスが表示される。



OK を押すとコンピュータの手が表示される。



コンピュータの応答を早くするには色々あるが、例えば、

```
if (((row-1>=0)&&(col+1<8)) && (board[row-1][col+1]==enemyPoint)) {
    flag = false;
    for (int i=2; (row-i>=0)&&(col+i<8); i++) {
        if (board[row-i][col+i]==playerPoint) {
```

```
                flag = true;
                break;
            } else if (board[row-i][col+i]==None)
                break;
        }
        if (flag) {
            k = 1;
            while (board[row-k][col+k]==enemyPoint) {
                board[row-k][col+k] = playerPoint;
                k++;
            }
        }
    }
```

の if 文で ((row-1>=0)&&(col+1<8)) の判定が必要ないように board[][]　をひとまわり大きくしておくとか、

```
                    for (int i=0; i<8; i++)
                        for (int j=0; j<8; j++)
                            board[i][j] = oldboard.board[i][j];
```

を高速に実行するために board[][]　を二次元の配列ではなく一次元の配列にし、memcpy() 関数を使って一気にコピーするとか、ものの本には書いてありますが、実際にやってみるとそれ程の効果はないです。

それよりは i3 とか i5 とか i7 の CPU が最近のパソコンには使われているので、

```
        for (int k=0; k<MAXUCTLOOP; k++) {
            ........
        }
```

の部分をマルチスレッドにすれば高速化が図れるはずですが、Windows フォームアプリケーションのマルチスレッドプログラミングの適切な情報を得るのが困難で非常に難しいです。

注意：　上のプログラムはモンテカルロ法の理論の断片を元に強引にプログラミングしましたが、最近、松原仁編　美添一樹・山下宏著「コンピュータ囲碁　モンテカルロ法の理論と実践」共立出版　２０１２年１１月　２８００円　という本が出版されました。第１０章　**UCT** で探索するプログラムにある配列を使ったプログラムを修正するほうが上で述べたプログラムより簡単で理解しやすいみたいです。自分でやってみてください。オセロは手の数が少なく、終局も分かりやすいので、単純なモンテカルロ法でも強くなると思って作ってみましたが期待したほどではなかったです。私が使っている定石が適切なのかもよくわかりません。興味があれば自分で作り直してください。

マルチスレッドとネットワークのプログラミングは次のようになります。Windows フォームアプリケーションで作ろうとするとデバッグが難しいのでまずはコンソールアプリケーションとして作ります。

```
#include <iostream>
#include <stdio.h>
#include <WinSock2.h>
#include <WS2tcpip.h>
#include <io.h>
#include <Windows.h>

#include <vector>

/* 「Ws2_32.lib」ファイルとリンクする */
#pragma comment (lib, "Ws2_32.lib")

using namespace std;

typedef enum {Black, White, None, Wall} KIND;
typedef enum {BLACK, WHITE, EMPTY} WHO;
typedef enum {RULE, UCT} TACTICS;
typedef enum {NORMAL, MAIN, SUB, HALF} ROTATION;

/* Period parameters */
#define N 624
#define M 397
#define MATRIX_A 0x9908b0dfUL   /* constant vector a */
#define UPPER_MASK 0x80000000UL /* most significant w-r bits */
#define LOWER_MASK 0x7fffffffUL /* least significant r bits */

static unsigned long mt[N]; /* the array for the state vector  */
static int mti=N+1; /* mti==N+1 means mt[N] is not initialized */

/* initializes mt[N] with a seed */
void init_genrand(unsigned long s)
{
    mt[0]= s & 0xffffffffUL;
    for (mti=1; mti<N; mti++) {
        mt[mti] =
    (1812433253UL * (mt[mti-1] ^ (mt[mti-1] >> 30)) + mti);
        /* See Knuth TAOCP Vol2. 3rd Ed. P.106 for multiplier. */
        /* In the previous versions, MSBs of the seed affect   */
        /* only MSBs of the array mt[].                        */
        /* 2002/01/09 modified by Makoto Matsumoto             */
        mt[mti] &= 0xffffffffUL;
        /* for >32 bit machines */
    }
```

```
}

/* initialize by an array with array-length */
/* init_key is the array for initializing keys */
/* key_length is its length */
/* slight change for C++, 2004/2/26 */
void init_by_array(unsigned long init_key[], int key_length)
{
    int i, j, k;
    init_genrand(19650218UL);
    i=1; j=0;
    k = (N>key_length ? N : key_length);
    for (; k; k--) {
        mt[i] = (mt[i] ^ ((mt[i-1] ^ (mt[i-1] >> 30)) * 1664525UL))
          + init_key[j] + j; /* non linear */
        mt[i] &= 0xffffffffUL; /* for WORDSIZE > 32 machines */
        i++; j++;
        if (i>=N) { mt[0] = mt[N-1]; i=1; }
        if (j>=key_length) j=0;
    }
    for (k=N-1; k; k--) {
        mt[i] = (mt[i] ^ ((mt[i-1] ^ (mt[i-1] >> 30)) * 1566083941UL))
          - i; /* non linear */
        mt[i] &= 0xffffffffUL; /* for WORDSIZE > 32 machines */
        i++;
        if (i>=N) { mt[0] = mt[N-1]; i=1; }
    }

    mt[0] = 0x80000000UL; /* MSB is 1; assuring non-zero initial array */
}

/* generates a random number on [0,0xffffffff]-interval */
unsigned long genrand_int32(void)
{
    unsigned long y;
    static unsigned long mag01[2]={0x0UL, MATRIX_A};
    /* mag01[x] = x * MATRIX_A  for x=0,1 */

    if (mti >= N) { /* generate N words at one time */
        int kk;

        if (mti == N+1)   /* if init_genrand() has not been called, */
            init_genrand(5489UL); /* a default initial seed is used */
```

```
        for (kk=0;kk<N-M;kk++) {
            y = (mt[kk]&UPPER_MASK)|(mt[kk+1]&LOWER_MASK);
            mt[kk] = mt[kk+M] ^ (y >> 1) ^ mag01[y & 0x1UL];
        }
        for (;kk<N-1;kk++) {
            y = (mt[kk]&UPPER_MASK)|(mt[kk+1]&LOWER_MASK);
            mt[kk] = mt[kk+(M-N)] ^ (y >> 1) ^ mag01[y & 0x1UL];
        }
        y = (mt[N-1]&UPPER_MASK)|(mt[0]&LOWER_MASK);
        mt[N-1] = mt[M-1] ^ (y >> 1) ^ mag01[y & 0x1UL];

        mti = 0;
    }

    y = mt[mti++];

    /* Tempering */
    y ^= (y >> 11);
    y ^= (y << 7) & 0x9d2c5680UL;
    y ^= (y << 15) & 0xefc60000UL;
    y ^= (y >> 18);

    return y;
}

struct CPoint {
    int row, col;
    CPoint(){row=-1; col=-1;}
    CPoint(int r, int c){row=r; col=c;}
};

struct LOG {
    string te[100];
    int n_te;
    LOG() {n_te = 0;}
};

struct J_Tree {
    J_Tree *child;
    J_Tree *sibling;
    int row;
    int col;
```

```
    J_Tree() {row = -1; col = -1; child = 0; sibling = 0; }
    J_Tree(int x, int y) {row = y; col = x; child = 0; sibling = 0; }
};

WHO who;
int number;
bool PlayerFirstP;
bool UserPlayP;
bool PassP;
LOG loglist;
TACTICS Tactics;
ROTATION direction;
J_Tree *rootBW;
J_Tree *currentBW;

string IntToAlphabet(int n)
{
    string S;

    switch (n) {
    case 1:
        S = "a"; break;
    case 2:
        S = "b"; break;
    case 3:
        S = "c"; break;
    case 4:
        S = "d"; break;
    case 5:
        S = "e"; break;
    case 6:
        S = "f"; break;
    case 7:
        S = "g"; break;
    case 8:
        S = "h"; break;
    }
    return S;
}

void Set_J_TreeB()
{
    J_Tree *p;
```

```
string joseki[98] = {
    "f5f4e3f6d3e2f2c5f1c4e6f3c3d7",
    "f5f4e3f6d3c5d6c4e6c7d7",
    "f5f6e6d6c5e3d3g5e2b5",
    "f5f6e6d6c5e3d3g5f3b5",
    "f5f6e6d6c5e3d3g5d7c7",
    "f5f6e6d6c5e3d3g5d7c6",
    "f5f6e6d6c5e3d3c4c6b5",
    "f5f6e6d6c5e3d3c4b5b6",
    "f5f6e6d6c5e3e7c7d7c6",
    "f5f6e6d6c5e3e7c6f7g5",
    "f5f6e6d6c5e3e7f4f7f8",
    "f5f6e6d6c5e3e7c4f4g6",
    "f5f6e6d6c5e3e7g5f7f8",
    "f5f6e6d6c5f4d7c4c3",
    "f5f6e6d6c5f4e7c4d3",
    "f5f6e6d6c5f4g5g4d3",
    "f5f6e6d6c5f4e3c6d7",
    "f5f6e6d6c5f4d3b5g4",
    "f5f6e6d6c4g5c6c5b6b5a6",
    "f5f6e6d6c4f4c6c5b6c3e3",
    "f5f6e6d6c4g4c5f4g5e3d3",
    "f5f6e6d6c6f4d7c8e8c7f7",
    "f5f6e6d6c6e3f3c5e7g5g4",
    "f5f6e6d6c6e3d3c5c4b5d7",
    "f5f6e6d6c3f4c6d3e3d2f3",
    "f5f6e6d6c3f4c6d3e3f2d7",
    "f5f6e6d6c3f4c6e3d7c7g5",
    "f5f6e6d6c3f4c6e3g5g4c7",
    "f5f6e6d6c3g4c6f4e7d7c5",
    "f5f6e6d6c3g4c6f4g5e3d3",
    "f5f6e6d6c3g5c6c5c4b5b6",
    "f5f6e6d6c3d3c4f3c5c6d2",
    "f5f6e6d6e7g5g6e8c4f3g4",
    "f5f6e6d6e7g5g6e8d7f7c6",
    "f5f6e6d6e7g5g6e8h5e3d7",
    "f5f6e6d6e7g5g6e8h4g4c5",
    "f5f6e6d6e7g5g6e3c6d7c7",
    "f5f6e6d6e7g5g6e3h5f8d8",
    "f5f6e6d6e7g5g6f7f8d8",
    "f5f6e6d6e7g5c5d7c4f3",
    "f5f6e6d6e7g5c5f7c4e3",
    "f5f6e6d6e7g5c5f7f4d3e3",
```

"f5f6e6d6e7g5c5f7c6f4g6",
"f5f6e6d6e7g5c5f7g6f4e8",
"f5f6e6d6e7g5c5f4c4d7",
"f5f6e6d6e7g5c5c7c6f4",
"f5f6e6d6e7f4g6h6g5f7",
"f5f6e6d6e7f4g5g6c5",
"f5f6e6d6e7f4g4g5e3",
"f5f6e6d6e7f4e3d7c4",
"f5f6e6d6e7f4c4c5e3",
"f5f6e6d6e7f4c5d7g5",
"f5f6e6d6e7f4c6f7g6",
"f5f6e6d6e7f7d7g5c5c6c7",
"f5f6e6d6e7f8c5e3",
"f5f6e6d6e7d8c5e3",
"f5f6e6d6e7f3c6f7d7f4e3",
"f5f6e6d6f7e3d7e7c6c5",
"f5f6e6d6f7f4d7g6",
"f5f6e6d6f7f4c5c6",
"f5f6e6d6f7f3e7f4e3g6g5",
"f5f6e6d6d7g5g4e7c5e3",
"f5f6e6d6c7c6f7e3f4d7",
"f5d6c3d3c4f4c5b3c2e6c6b4b5d2e3",
"f5d6c3d3c4f4c5b3c2e6c6b4b5d2a3",
"f5d6c3d3c4f4c5b3c2e6c6b4b5d2f7",
"f5d6c3d3c4f4c5b3c2e6c6b4b5d2c7",
"f5d6c3d3c4f4c5b3c2b4c6d2e6b5a5",
"f5d6c3d3c4f4c5b3c2e3d2c6b4a3g3",
"f5d6c3d3c4f4c5b4c6e6a3b5e3",
"f5d6c3d3c4f4f6g5e6c5f3b5e3",
"f5d6c3d3c4f4f6g5g6e3h5",
"f5d6c3d3c4f4f6g5c6e3d7",
"f5d6c3d3c4f4f6g5e3f3g6",
"f5d6c3d3c4f4f6g5f3g6h4",
"f5d6c3d3c4f4f6f3e6e7f7c5b6",
"f5d6c3d3c4f4f6f3e6e7d7g6e8c5c6",
"f5d6c3d3c4f4e6f6e3c5g5g3b6h6c6",
"f5d6c3d3c4f4e6b3e2e3f3c5b4a3f2",
"f5d6c3d3c4f4e3f6c6c5d7e7e6",
"f5d6c3d3c4f4f3e3c6c5f6e6b6",
"f5d6c3d3c4f4c6e3d7f6d2c5b4",
"f5d6c3d3c4f4c6c5e3f3b6b5g4",
"f5d6c3d3c4b3c6b6d7e8c2c1a3",
"f5d6c4d3c5f4e3f3c2f6d7",

```
    "f5d6c4d3e6f4e3f3c6f6g5",
    "f5d6c4g5c6c5b6c3e3b5",
    "f5d6c3f4f6d3c6b6c2d2",
    "f5d6c3g5c6c5c4b6f6f4",
    "f5d6c5f4e3c6d3f6e6d7g3c4g5",
    "f5d6c5f4e3c6d3f6e6d7e7c7c4",
    "f5d6c5f4e3c6d3f3e6f7g4c3c2",
    "f5d6c5f4e3c6d3g5f6f3g6c4c3",
    "f5d6c5f4e3c6e6f7d7e8f3f6",
    "f5d6c5f4e3c6e6f6g3c4",
    "f5d6c5f4e3d3f3e2c4c3f2b3e1",
    "f5d6c5f4e3d3e6g5c4c3d2f6c2",
    "f5d6c5f6e6f4c6e7f8c4c3"
};

rootBW = 0;
for(int i=0; i<98; i++) {
    string str = joseki[i];
    int len = str.length();
    p = rootBW;
    for (int i=0; i<len/2; i++) {
        int col = str[2*i]-'a'+1;
        int row = str[2*i+1]-'1'+1;
        if (p == 0) {
            rootBW = new J_Tree(col, row);
            p = rootBW;
            continue;
        } else {
            if (p->col == col && p->row == row) {
                continue;
            }
            if (p->child == 0) {
                p->child = new J_Tree(col, row);
                p = p->child;
                continue;
            } else if (p->child->col == col && p->child->row == row) {
                p = p->child;
                continue;
            } else {
                bool flag = false;
                p = p->child;
                while (p->sibling != 0) {
                    if (p->sibling->col == col && p->sibling->row == row) {
```

```
                            p = p->sibling;
                            flag = true;
                            break;
                        }
                        p = p->sibling;
                    }
                    if (!flag) {
                        p->sibling = new J_Tree(col, row);
                        p = p->sibling;
                    }
                }
            }
        }
    }
}

#define Max  99999
#define Min -99999
#define uct_max 100
#define max_depth 12
#define MAXUCTLOOP 100000
#define THREAD_NUM 8

bool OnePlayOutPassP;

struct UCT_Tree {
    double UCB;
    double X;
    int row;
    int col;
    WHO who;
    int n;
    int CN;
    UCT_Tree * parent;
    UCT_Tree * child;
    UCT_Tree * next;
    UCT_Tree(){row=-1; col=-1; n=0; CN=0; parent=0; child=0; next=0;}
};

struct Board {
    KIND board[160];
    UCT_Tree *tree;
    Board();
```

```
    void add(int row, int col, WHO who);
    int Calculate(WHO who);
    int Calculate2(WHO who);
    bool LegalPointP(int row, int col, WHO who);
    void GenAllMoves(vector<CPoint> &movelist, WHO who);
    CPoint ComputerMove(WHO who, TACTICS tactics);
    void delete_UCT_Tree(UCT_Tree *tree);
    int OnePlayOut(CPoint pt, WHO who, WHO origin);
    double change_UCT_Tree(UCT_Tree *tree, bool PositiveP);
    void showBoard();
    CPoint HumanMove(WHO who);
};

CPoint Board::HumanMove(WHO who)
{
    CPoint move;
    int row, col;

    do {
        cout << "col row : ";
        cin >> col >> row;
        if (col == -1)
            return move;
    } while (!LegalPointP(row, col, who));
    move.row = row;
    move.col = col;
    return move;
}

string IntToDisp(int n)
{
    string str;

    switch (n) {
    case 0:
        str = "０"; break;
    case 1:
        str = "１"; break;
    case 2:
        str = "２"; break;
    case 3:
        str = "３"; break;
    case 4:
```

```
        str = "４"; break;
    case 5:
        str = "５"; break;
    case 6:
        str = "６"; break;
    case 7:
        str = "７"; break;
    case 8:
        str = "８"; break;
    }
    return str;
}

void Board::showBoard()
{
    for (int col=0; col<9; col++)
        if (col == 0)
            cout << "  ";
        else
            cout << IntToDisp(col).c_str();
    cout << "\n";
    for (int row=1; row <9; row++) {
        cout << IntToDisp(row).c_str();
        for (int col=1; col<9; col++) {
            if (board[16*row+col]==Black)
                cout << "●";
            else if (board[16*row+col]==White)
                cout << "○";
            else
                cout << "  ";
        }
        cout << "\n";
    }
}

Board::Board()
{
    for (int i=0; i<160; i++)
        board[i] = None;
    for (int k=0; k<10; k+=9)
        for (int i=0; i<10; i++)
            board[16*k+i] = Wall;
    for (int k=0; k<10; k+=9)
```

```
        for (int i=0; i<10; i++)
            board[16*i+k] = Wall;
    board[4*16+4] = White;
    board[4*16+5] = Black;
    board[5*16+4] = Black;
    board[5*16+5] = White;
    tree = 0;
}

void Board::add(int row, int col, WHO player)
{
    KIND playerPoint, enemyPoint;
    bool flag;
    int k;

    if (player == BLACK) {
        playerPoint = Black;
        enemyPoint = White;
    } else {
        playerPoint = White;
        enemyPoint = Black;
    }
    board[16*row+col] = playerPoint;
    if (board[16*row+col-1]==enemyPoint) {
        flag = false;
        for (int i=2; col-i > 0; i++) {
            if (board[16*row+col-i]==playerPoint) {
                flag = true;
                break;
            } else if (board[16*row+col-i]==None)
                break;
        }
        if (flag) {
            k = 1;
            while (board[16*row+col-k]==enemyPoint) {
                board[16*row+col-k] = playerPoint;
                k++;
            }
        }
    }
    if (board[16*row+col+1]==enemyPoint) {
        flag = false;
        for (int i=2; col+i < 9; i++) {
```

```
            if (board[16*row+col+i]==playerPoint) {
                flag = true;
                break;
            } else if (board[16*row+col+i]==None)
                break;
        }
        if (flag) {
            k = 1;
            while (board[16*row+col+k]==enemyPoint) {
                board[16*row+col+k] = playerPoint;
                k++;
            }
        }
    }
    if (board[16*(row-1)+col]==enemyPoint) {
        flag = false;
        for (int i=2; row-i > 0; i++) {
            if (board[16*(row-i)+col]==playerPoint) {
                flag = true;
                break;
            } else if (board[16*(row-i)+col]==None)
                break;
        }
        if (flag) {
            k = 1;
            while (board[16*(row-k)+col]==enemyPoint) {
                board[16*(row-k)+col] = playerPoint;
                k++;
            }
        }
    }
    if (board[16*(row+1)+col]==enemyPoint) {
        flag = false;
        for (int i=2; row+i < 9; i++) {
            if (board[16*(row+i)+col]==playerPoint) {
                flag = true;
                break;
            } else if (board[16*(row+i)+col]==None)
                break;
        }
        if (flag) {
            k = 1;
            while (board[16*(row+k)+col]==enemyPoint) {
```

```
            board[16*(row+k)+col] = playerPoint;
            k++;
        }
    }
}
if (board[16*(row-1)+col-1]==enemyPoint) {
    flag = false;
    for (int i=2; (row-i>0)&&(col-i>0); i++) {
        if (board[16*(row-i)+col-i]==playerPoint) {
            flag = true;
            break;
        } else if (board[16*(row-i)+col-i]==None)
            break;
    }
    if (flag) {
        k = 1;
        while (board[16*(row-k)+col-k]==enemyPoint) {
            board[16*(row-k)+col-k] = playerPoint;
            k++;
        }
    }
}
if (board[16*(row-1)+col+1]==enemyPoint) {
    flag = false;
    for (int i=2; (row-i>0)&&(col+i<9); i++) {
        if (board[16*(row-i)+col+i]==playerPoint) {
            flag = true;
            break;
        } else if (board[16*(row-i)+col+i]==None)
            break;
    }
    if (flag) {
        k = 1;
        while (board[16*(row-k)+col+k]==enemyPoint) {
            board[16*(row-k)+col+k] = playerPoint;
            k++;
        }
    }
}
if (board[16*(row+1)+col-1]==enemyPoint) {
    flag = false;
    for (int i=2; (row+i<9)&&(col-i>0); i++) {
        if (board[16*(row+i)+col-i]==playerPoint) {
```

```
            flag = true;
            break;
        } else if (board[16*(row+i)+col-i]==None)
            break;
    }
    if (flag) {
        k = 1;
        while (board[16*(row+k)+col-k]==enemyPoint) {
            board[16*(row+k)+col-k] = playerPoint;
            k++;
        }
    }
}
if (board[16*(row+1)+col+1]==enemyPoint) {
    flag = false;
    for (int i=2; (row+i<9)&&(col+i<9); i++) {
        if (board[16*(row+i)+col+i]==playerPoint) {
            flag = true;
            break;
        } else if (board[16*(row+i)+col+i]==None)
            break;
    }
    if (flag) {
        k = 1;
        while (board[16*(row+k)+col+k]==enemyPoint) {
            board[16*(row+k)+col+k] = playerPoint;
            k++;
        }
    }
}
if (board[16*(row-1)+col+1]==enemyPoint) {
    flag = false;
    for (int i=2; (row-i>0)&&(col+i<9); i++) {
        if (board[16*(row-i)+col+i]==playerPoint) {
            flag = true;
            break;
        } else if (board[16*(row-i)+col+i]==None)
            break;
    }
    if (flag) {
        k = 1;
        while (board[16*(row-k)+col+k]==enemyPoint) {
            board[16*(row-k)+col+k] = playerPoint;
```

```
                k++;
            }
        }
    }
}

int Board::Calculate(WHO who)
{
    KIND playerPoint, enemyPoint;

    if (who == BLACK) {
        playerPoint = Black;
        enemyPoint = White;
    } else {
        playerPoint = White;
        enemyPoint = Black;
    }
    int Pnum=0;
    for (int row=1; row<9; row++)
        for (int col=1; col<9; col++) {
            if (board[16*row+col] == playerPoint)
                Pnum ++;
        }
    return Pnum;
}

int Board::Calculate2(WHO who)
{
    KIND playerPoint, enemyPoint;

    if (who == BLACK) {
        playerPoint = Black;
        enemyPoint = White;
    } else {
        playerPoint = White;
        enemyPoint = Black;
    }
    int Pnum=0, Enum=0;
    for (int row=1; row<9; row++)
        for (int col=1; col<9; col++) {
            if (board[16*row+col] == playerPoint)
                Pnum++;
            else if (board[16*row+col] == enemyPoint)
```

```
                Enum++;
        }
    return Pnum-Enum;
}

bool Board::LegalPointP(int row, int col, WHO player)
{
    bool flag;
    KIND playerPoint, enemyPoint;

    if (board[16*row+col] != None)
        return false;
    if (player == BLACK) {
        playerPoint = Black;
        enemyPoint = White;
    } else {
        playerPoint = White;
        enemyPoint = Black;
    }
    if (board[16*row+col-1]==enemyPoint) {
        flag = false;
        for (int i=2; col-i > 0; i++) {
            if (board[16*row+col-i]==playerPoint) {
                flag = true;
                break;
            } else if (board[16*row+col-i]==None)
                break;
        }
        if (flag)
            return true;
    }
    if (board[16*row+col+1]==enemyPoint) {
        flag = false;
        for (int i=2; col+i < 9; i++) {
            if (board[16*row+col+i]==playerPoint) {
                flag = true;
                break;
            } else if (board[16*row+col+i]==None)
                break;
        }
        if (flag)
            return true;
    }
```

```
if (board[16*(row-1)+col]==enemyPoint) {
    flag = false;
    for (int i=2; row-i > 0; i++) {
        if (board[16*(row-i)+col]==playerPoint) {
            flag = true;
            break;
        } else if (board[16*(row-i)+col]==None)
            break;
    }
    if (flag) {
        return true;
    }
}
if (board[16*(row+1)+col]==enemyPoint) {
    flag = false;
    for (int i=2; row+i < 9; i++) {
        if (board[16*(row+i)+col]==playerPoint) {
            flag = true;
            break;
        } else if (board[16*(row+i)+col]==None)
            break;
    }
    if (flag)
        return true;
}
if (board[16*(row-1)+col-1]==enemyPoint) {
    flag = false;
    for (int i=2; (row-i>0)&&(col-i>0); i++) {
        if (board[16*(row-i)+col-i]==playerPoint) {
            flag = true;
            break;
        } else if (board[16*(row-i)+col-i]==None)
            break;
    }
    if (flag) {
        return true;
    }
}
if (board[16*(row-1)+col+1]==enemyPoint) {
    flag = false;
    for (int i=2; (row-i>0)&&(col+i<9); i++) {
        if (board[16*(row-i)+col+i]==playerPoint) {
            flag = true;
```

```
            break;
        } else if (board[16*(row-i)+col+i]==None)
            break;
    }
    if (flag)
        return true;
}
if (board[16*(row+1)+col-1]==enemyPoint) {
    flag = false;
    for (int i=2; (row+i<9)&&(col-i>0); i++) {
        if (board[16*(row+i)+col-i]==playerPoint) {
            flag = true;
            break;
        } else if (board[16*(row+i)+col-i]==None)
            break;
    }
    if (flag)
        return true;
}
if (board[16*(row+1)+col+1]==enemyPoint) {
    flag = false;
    for (int i=2; (row+i<9)&&(col+i<9); i++) {
        if (board[16*(row+i)+col+i]==playerPoint) {
            flag = true;
            break;
        } else if (board[16*(row+i)+col+i]==None)
            break;
    }
    if (flag)
        return true;
}
if (board[16*(row-1)+col+1]==enemyPoint) {
    flag = false;
    for (int i=2; (row-i>0)&&(col+i<9); i++) {
        if (board[16*(row-i)+col+i]==playerPoint) {
            flag = true;
            break;
        } else if (board[16*(row-i)+col+i]==None)
            break;
    }
    if (flag)
        return true;
}
```

```
    return false;
}

void Board::GenAllMoves(vector<CPoint> &movelist, WHO who)
{
    for (int row=1; row<9; row++)
        for (int col=1; col<9; col++)
            if (LegalPointP(row, col, who))
                movelist.push_back(CPoint(row, col));
}

void Board::delete_UCT_Tree(UCT_Tree *tree)
{
    UCT_Tree *p;
    while (tree != 0) {
        p = tree;
        tree = tree->next;
        if (p->child != 0)
            delete_UCT_Tree(p->child);
        delete p;
    }
}

struct thread_arg {
    Board game;
    WHO who;
};

int cand[160];
HANDLE Mutex;

int Board::OnePlayOut(CPoint pt, WHO who, WHO origin)
{
    vector<CPoint> movelist;
    CPoint move;
    int result;

    add(pt.row, pt.col, who);
    bool OnePlayOutPassP = false;

GG: if (who == BLACK) {
        who = WHITE;
    } else {
```

```
        who = BLACK;
    }
    movelist.clear();
    GenAllMoves(movelist, who);
    if (movelist.size() == 0) {
        if (OnePlayOutPassP) {
            result = Calculate2(origin);
            if (result > 0)
                result = 1;
            else if (result < 0)
                result = -1;
            return result;
        } else {
            OnePlayOutPassP = true;
            goto GG;
        }
    }
    int ind = genrand_int32() % movelist.size();
    move = movelist[ind];
    result = OnePlayOut(move, who, origin);
    return result;
}

void thread_func(void *arg)
{
    thread_arg *targ = (thread_arg *)arg;
    Board mygame = targ->game;
    WHO who = targ->who;
    vector<CPoint> movelist;
    UCT_Tree *pp, *maxpp;
    WHO currentWho = who;
    mygame.tree = 0;

    movelist.clear();
    mygame.GenAllMoves(movelist, who);
    for (int i=0; i<movelist.size(); i++) {
        pp = new UCT_Tree();
        pp->row = movelist[i].row;
        pp->col = movelist[i].col;
        pp->who = who;
        pp->UCB = 500.0 + genrand_int32() % 20;
        pp->X = 0.0;
        pp->n = 0;
```

```
        pp->CN = 0;
        pp->parent = 0;
        pp->child = 0;
        pp->next = mygame.tree;
        mygame.tree = pp;
    }
    double MaxUCB;
    int depth;
    for (int k=0; k<MAXUCTLOOP; k++) {
        Board oldboard;
        memcpy(oldboard.board, mygame.board, 160*sizeof(KIND));
        pp = mygame.tree;
        depth = 0;
LOOPA:  MaxUCB = pp->UCB;
        pp->CN++;
        maxpp = pp;
        depth++;
        pp = pp->next;
        while (pp != 0) {
            if (currentWho == pp->who) {
                if (pp->UCB > MaxUCB) {
                    MaxUCB = pp->UCB;
                    maxpp = pp;
                }
            } else {
                if (pp->UCB < MaxUCB) {
                    MaxUCB = pp->UCB;
                    maxpp = pp;
                }
            }
            pp->CN++;
            pp = pp->next;
        }
        if (maxpp->child != 0) {
            mygame.add(maxpp->row, maxpp->col, maxpp->who);
            maxpp->n++;
            pp = maxpp->child;
            goto LOOPA;
        } else { // maxpp->child == 0
            maxpp->n++;
            if (depth < max_depth && maxpp->n > uct_max) {
                // 新しい子供を作る
                if (maxpp->who == BLACK)
```

```
    who = WHITE;
else
    who = BLACK;
mygame.add(maxpp->row, maxpp->col, maxpp->who);
movelist.clear();
mygame.GenAllMoves(movelist, who);
if (movelist.size() == 0) { // leaf or PASS
    // leaf?
    int count = 0;
    for (int i=1; i<9; i++) {
        for (int j=1; j<9; j++) {
            if (mygame.board[16*i+j] == None) {
                count++;
            }
        }
    }
    if (count > 0) {
        goto HHH;
    }
    int result = mygame.Calculate2(currentWho);
    if (result > 0)
        result = 1;
    else if (result < 0)
        result = -1;
    // UCB の更新 と pp->X の修正
    maxpp->X = (maxpp->X*(maxpp->n-1)+result)/(maxpp->n);
    if (maxpp->parent == 0) {
        pp = mygame.tree;
    } else {
        pp = maxpp->parent;
        pp = pp->child;
    }
    while (pp != 0) {
        if (pp->n > 0) {
            if (currentWho == pp->who) {
                pp->UCB = pp->X+sqrt(2.0*log((double)pp->CN)/(pp->n));
            } else {
                pp->UCB = pp->X-sqrt(2.0*log((double)pp->CN)/(pp->n));
            }
        }
        pp = pp->next;
    }
    // 親の pp->UCB と pp->X の修正
```

```
        UCT_Tree *qq = maxpp->parent;
        while (qq != 0) {
            qq->X = (qq->X*(qq->n-1)+result)/(qq->n);
            if (qq->parent == 0) {
                pp = mygame.tree;
            } else {
                pp = qq->parent;
                pp = pp->child;
            }
            while (pp != 0) {
                if (currentWho == pp->who) {
                    if (pp->n > 0)
                        pp->UCB = pp->X+sqrt(2.0*log((double)pp->CN)/(pp->n));
                } else {
                    if (pp->n > 0)
                        pp->UCB = pp->X-sqrt(2.0*log((double)pp->CN)/(pp->n));
                }
                pp = pp->next;
            }
            qq = qq->parent;
        }
        memcpy(mygame.board, oldboard.board, 160*sizeof(KIND));
        continue;
    } else { // non leaf
        for (int i=0; i<movelist.size(); i++) {
            pp = new UCT_Tree();
            pp->row = movelist[i].row;
            pp->col = movelist[i].col;
            pp->who = (maxpp->who==BLACK)?WHITE:BLACK;
            if (currentWho == pp->who) {
                pp->UCB = 500.0 + genrand_int32() % 20;
            } else {
                pp->UCB = -500.0 + genrand_int32() % 20;
            }
            pp->X = 0.0;
            pp->n = 0;
            pp->CN = 1;
            pp->parent = maxpp;
            pp->child = 0;
            pp->next = maxpp->child;
            maxpp->child = pp;
        }
        pp = maxpp->child;
```

```
MaxUCB = pp->UCB;
maxpp = pp;
pp = pp->next;
while (pp != 0) {
    if (currentWho == pp->who) {
        if (pp->UCB > MaxUCB) {
            MaxUCB = pp->UCB;
            maxpp = pp;
        }
    } else {
        if (pp->UCB < MaxUCB) {
            MaxUCB = pp->UCB;
            maxpp = pp;
        }
    }
    pp = pp->next;
}
CPoint C;
C.row = maxpp->row;
C.col = maxpp->col;
OnePlayOutPassP = false;
int result = mygame.OnePlayOut(C, maxpp->who, currentWho);
// UCB と pp->X の修正
maxpp->n++;
maxpp->X = (maxpp->X*(maxpp->n-1)+result)/(maxpp->n);
if (maxpp->parent == 0) {
    pp = mygame.tree;
} else {
    pp = maxpp->parent;
    pp = pp->child;
}
while (pp != 0) {
    if (pp->n > 0) {
        if (currentWho == pp->who) {
            pp->UCB = pp->X+sqrt(2.0*log((double)pp->CN)/(pp->n));
        } else {
            pp->UCB = pp->X-sqrt(2.0*log((double)pp->CN)/(pp->n));
        }
    }
    pp = pp->next;
}
// 親の pp->UCB と pp->X の修正
UCT_Tree *qq = maxpp->parent;
```

```
                    while (qq != 0) {
                        qq->X = (qq->X*(qq->n-1)+result)/(qq->n);
                        if (qq->parent == 0) {
                            pp = mygame.tree;
                        } else {
                            pp = qq->parent;
                            pp = pp->child;
                        }
                        while (pp != 0) {
                            if (currentWho == pp->who) {
                                if (pp->n > 0)
                                    pp->UCB = pp->X+sqrt(2.0*log((double)pp->CN)/(pp->n));
                            } else {
                                if (pp->n > 0)
                                    pp->UCB = pp->X-sqrt(2.0*log((double)pp->CN)/(pp->n));
                            }
                            pp = pp->next;
                        }
                        qq = qq->parent;
                    }
                    memcpy(mygame.board, oldboard.board, 160*sizeof(KIND));
                    continue;
                }
            } else {
                // 新しい子供は作らない  maxpp->child == 0
                // maxpp->N  && maxpp->n は増やしている
HHH:            CPoint C;
                C.row = maxpp->row;
                C.col = maxpp->col;
                OnePlayOutPassP = false;
                int result = mygame.OnePlayOut(C, maxpp->who, currentWho);
                // UCB と pp->X の修正
                maxpp->X = (maxpp->X*(maxpp->n-1)+result)/(maxpp->n);
                if (maxpp->parent == 0) {
                    pp = mygame.tree;
                } else {
                    pp = maxpp->parent;
                    pp = pp->child;
                }
                while (pp != 0) {
                    if (pp->n > 0) {
                        if (currentWho == pp->who) {
                            pp->UCB = pp->X+sqrt(2.0*log((double)pp->CN)/(pp->n));
```

```
                    } else {
                        pp->UCB = pp->X-sqrt(2.0*log((double)pp->CN)/(pp->n));
                    }
                }
                pp = pp->next;
            }
            // 親の pp->UCB と pp->X の修正
            UCT_Tree *qq = maxpp->parent;
            while (qq != 0) {
                qq->X = (qq->X*(qq->n-1)+result)/(qq->n);
                if (qq->parent == 0) {
                    pp = mygame.tree;
                } else {
                    pp = qq->parent;
                    pp = pp->child;
                }
                while (pp != 0) {
                    if (currentWho == pp->who) {
                        if (pp->n > 0)
                            pp->UCB = pp->X+sqrt(2.0*log((double)pp->CN)/(pp->n));
                    } else {
                        if (pp->n > 0)
                            pp->UCB = pp->X-sqrt(2.0*log((double)pp->CN)/(pp->n));
                    }
                    pp = pp->next;
                }
                qq = qq->parent;
            }
            memcpy(mygame.board, oldboard.board, 160*sizeof(KIND));
            continue;
        }
    }
}
WaitForSingleObject(Mutex, 0);
pp = mygame.tree;
while (pp != 0) {
    int index = 16 * pp->row + pp->col;
    cand[index] += pp->n;
    pp = pp->next;
}
ReleaseMutex(Mutex);
mygame.delete_UCT_Tree(mygame.tree);
mygame.tree = 0;
```

```
}

CPoint Board::ComputerMove(WHO who, TACTICS tactics)
{
    vector<CPoint> movelist;
    CPoint move;
    char s[10];
    int col, row;

    if (tactics == RULE) {
        GenAllMoves(movelist, who);
        if (movelist.size() == 0)
            return move;
        return movelist[0];
    } else if (tactics == UCT) {
        if (who == BLACK && number == 0) {
            move.row = 5;
            move.col = 6;  // f5
            currentBW = rootBW;
            direction = NORMAL;
            return move;
        }
        if (who == WHITE && number == 1) {
            currentBW = rootBW;
            strcpy(s, loglist.te[loglist.n_te-1].c_str());
            col = s[0] - 'a' + 1;
            row = s[1];
            if (col == 6 && row == 5) {
                direction = NORMAL;
            } else if (col == 5 && row == 6) {
                direction = MAIN;
            } else if (col == 4 && row == 3) {
                direction = SUB;
            } else if (col == 3 && row == 4) {
                direction = HALF;
            }
        }
        strcpy(s, loglist.te[loglist.n_te-1].c_str());
        col = s[0] - 'a' + 1;
        row = s[1];
        if (direction == MAIN) {
            int junk = col;
            col = row;
```

```
    row = junk;
} else if (direction == SUB) {
    int junk = col;
    col = 9 - row;
    row = 9 - junk;
} else if (direction == HALF) {
    col = 9 - col;
    row = 9 - row;
}
if (currentBW != 0 && currentBW->child != 0) { // 定石を探す
    if (currentBW->col == col && currentBW->row == row) {
        currentBW = currentBW->child;
        if (currentBW != 0) {
            int count = 1;
            J_Tree *p = currentBW;
            while (p->sibling != 0) {
                count++;
                p = p->sibling;
            }
            int index = genrand_int32() % count;
            index++;
            if (index == 1) {
                if (direction == NORMAL) {
                    move.row = currentBW->row;
                    move.col = currentBW->col;
                } else if (direction == MAIN) {
                    move.row = currentBW->col;
                    move.col = currentBW->row;
                } else if (direction == SUB) {
                    move.row = 9-currentBW->col;
                    move.col = 9-currentBW->row;
                } else if (direction == HALF) {
                    move.row = 9-currentBW->row;
                    move.col = 9-currentBW->col;
                }
                return move;
            } else {
                p = currentBW;
                int cnt = 1;
                while (cnt < index) {
                    p = p->sibling;
                    cnt++;
                }
```

```
                currentBW = p;
                if (direction == NORMAL) {
                    move.row = currentBW->row;
                    move.col = currentBW->col;
                } else if (direction == MAIN) {
                    move.row = currentBW->col;
                    move.col = currentBW->row;
                } else if (direction == SUB) {
                    move.row = 9-currentBW->col;
                    move.col = 9-currentBW->row;
                } else if (direction == HALF) {
                    move.row = 9-currentBW->row;
                    move.col = 9-currentBW->col;
                }
                return move;
            }
        } else {
            currentBW = 0;
        }
    } else if (currentBW->child->col == col && currentBW->child->row == row) {
        currentBW = currentBW->child;
        if (currentBW->child != 0) {
            currentBW = currentBW->child;
            int count = 1;
            J_Tree *p = currentBW;
            while (p->sibling != 0) {
                count++;
                p = p->sibling;
            }
            int index = genrand_int32() % count;
            index++;
            if (index == 1) {
                if (direction == NORMAL) {
                    move.row = currentBW->row;
                    move.col = currentBW->col;
                } else if (direction == MAIN) {
                    move.row = currentBW->col;
                    move.col = currentBW->row;
                } else if (direction == SUB) {
                    move.row = 9-currentBW->col;
                    move.col = 9-currentBW->row;
                } else if (direction == HALF) {
                    move.row = 9-currentBW->row;
```

```
                move.col = 9-currentBW->col;
            }
            return move;
        } else {
            p = currentBW;
            int cnt=1;
            while (cnt < index) {
                p = p->sibling;
                cnt++;
            }
            currentBW = p;
            if (direction == NORMAL) {
                move.row = currentBW->row;
                move.col = currentBW->col;
            } else if (direction == MAIN) {
                move.row = currentBW->col;
                move.col = currentBW->row;
            } else if (direction == SUB) {
                move.row = 9-currentBW->col;
                move.col = 9-currentBW->row;
            } else if (direction == HALF) {
                move.row = 9-currentBW->row;
                move.col = 9-currentBW->col;
            }
            return move;
        }
    } else {
        currentBW = 0;
    }
} else {
    currentBW = currentBW->child->sibling;
    while (currentBW != 0) {
        if (currentBW->col == col && currentBW->row == row) {
            if (currentBW->child != 0) {
                currentBW = currentBW->child;
                int count = 1;
                J_Tree *p = currentBW;
                while (p->sibling != 0) {
                    count++;
                    p = p->sibling;
                }
                int index = genrand_int32() % count;
                index++;
```

```
            if (index == 1) {
                if (direction == NORMAL) {
                    move.row = currentBW->row;
                    move.col = currentBW->col;
                } else if (direction == MAIN) {
                    move.row = currentBW->col;
                    move.col = currentBW->row;
                } else if (direction == SUB) {
                    move.row = 9-currentBW->col;
                    move.col = 9-currentBW->row;
                } else if (direction == HALF) {
                    move.row = 9-currentBW->row;
                    move.col = 9-currentBW->col;
                }
                return move;
            } else {
                p = currentBW;
                int cnt = 1;
                while (cnt < index) {
                    p = p->sibling;
                    cnt++;
                }
                currentBW = p;
                if (direction == NORMAL) {
                    move.row = currentBW->row;
                    move.col = currentBW->col;
                } else if (direction == MAIN) {
                    move.row = currentBW->col;
                    move.col = currentBW->row;
                } else if (direction == SUB) {
                    move.row = 9-currentBW->col;
                    move.col = 9-currentBW->row;
                } else if (direction == HALF) {
                    move.row = 9-currentBW->row;
                    move.col = 9-currentBW->col;
                }
                return move;
            }
        } else {
            currentBW = 0;
            break;
        }
    }
```

```
                currentBW = currentBW->sibling;
            }
        }
    }
    // モンテカルロ法で手を探す
    movelist.clear();
    GenAllMoves(movelist, who);
    if (movelist.size() == 0)
        return move;
    if (movelist.size() == 1)
        return movelist[0];
    // スレッドの計算
    for (int k=0; k<160; k++)
        cand[k] = 0;
    HANDLE handle[THREAD_NUM];
    thread_arg targ[THREAD_NUM];
    Mutex = CreateMutex(NULL, TRUE, NULL);
    for (int i=0; i<THREAD_NUM; i++) {
        for (int k=0; k<160; k++)
            targ[i].game.board[k] = KIND(board[k]);
        targ[i].game.tree = 0;
        targ[i].who = who;
        handle[i] = CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)thread_func,
                                 (void *)&targ[i], 0, NULL);
    }
    WaitForMultipleObjects(THREAD_NUM, handle, TRUE, INFINITE);
    int candidate = 0;
    int maxvalue = 0;
    for (int k=0; k<160; k++)
        if (cand[k] > maxvalue) {
            maxvalue = cand[k];
            candidate = k;
        }
    move.row = candidate / 16;
    move.col = candidate % 16;
    CloseHandle(Mutex);
    return move;
  }
}

Board *game;

int main(int argc, char *argv[])
```

```
{
    char c;
    vector<CPoint> movelist;
    CPoint move;
    WSADATA wsaData;
    SOCKET sock, fd, ret;
    struct sockaddr_in local, addr;
    struct sockaddr_in client;
    int isExit = 0;

    /* Winsock を初期化する */
    WSAStartup(MAKEWORD(2, 0), &wsaData);
    /* ソケットを作成する */
    sock = socket(PF_INET, SOCK_STREAM, 0);

    if (sock == -1)
        return 0;   /* 失敗したので終了 */

    cout << "先手２＝ y or n ";
    cin >> c;

    if (c == 'y') {
        PlayerFirstP = true;
        PassP = false;
        who = BLACK;
        /* 接続待ちを行うために、バインドする */
        memset(&local, 0, sizeof(local));
        local.sin_family = PF_INET;
        local.sin_port = htons(50000);
        local.sin_addr.s_addr = htonl(INADDR_ANY);
    } else {
        PlayerFirstP = false;
        PassP = false;
        who = WHITE;
        /* 接続先の設定 */
        memset(&addr, 0, sizeof(addr));
        addr.sin_family = PF_INET;
        addr.sin_port = htons(50000);
        addr.sin_addr.s_addr = inet_addr("127.0.0.1");
    }
    game = new Board();
    Set_J_TreeB();
    number = 0;
```

```
if (PlayerFirstP) {
    /* サーバー側 */
    if (bind(sock, (const struct sockaddr *)&local, sizeof(local)) == 0) {
        /* バインドできたので、接続待ち */
        if (listen(sock, 10) == 0) {
            socklen_t len = sizeof(client);
            fd = accept(sock, (struct sockaddr *)&client, &len);
            char buf[256];
            if (fd < 0) {
                /* エラー発生 */
                isExit = 1;
            } else {
                /* クライアントからの接続があったので通信する */
                int buflen;
                /* クライアントにメッセージを送る */
                strcpy_s(buf, sizeof(buf), "START\n");
                send(fd, buf, strlen(buf), 0);
                /* クライアントからの入力待ち */
                buflen = recv(fd, buf, strlen(buf), 0);
                /* 文字列として扱うために NULL 文字を追加 */
                buf[ buflen] = 0;
                /* 入力された文字列が「OK」なら OK*/
                if (strcmp(buf, "OK") == 0) {
                    printf("Received: %s\n", buf);
                } else {
                    /* それ以外のときは、終了 */
                    isExit = 1;
                }
            }
            /* クライアントとの接続を閉じる */
            closesocket(fd);
            /* 接続待ちを行うループ */
            while (!isExit) {
                socklen_t len = sizeof(client);
                fd = accept(sock, (struct sockaddr *)&client, &len);
                if (fd < 0) {
                    /* エラー発生 */
                    isExit = 1;
                } else {
                    /* クライアントからの接続があったので通信する */
                    /* 手を送信する */
                    who = BLACK;
```

```
movelist.clear();
game->GenAllMoves(movelist, who);
if (movelist.size() == 0) {
    if (PassP) {
        PassP = true;
        loglist.te[loglist.n_te++] = "PS";
        number++;
        /* クライアントにメッセージを送る */
        move.row = move.col = 0;
        sprintf(buf, "%d %d\n", move.row, move.col);
        send(fd, buf, strlen(buf), 0);
        printf("● A PS\n");
        game->showBoard();
        closesocket(fd);
        break;
    } else {
        PassP = true;
        loglist.te[loglist.n_te++] = "PS";
        number++;
        /* クライアントにメッセージを送る */
        move.row = move.col = 0;
        sprintf(buf, "%d %d\n", move.row, move.col);
        send(fd, buf, strlen(buf), 0);
        printf("● B PS buf=%s\n", buf);
        game->showBoard();
    }
} else {
    move = game->ComputerMove(who, UCT);
    game->add(move.row, move.col, who);
    loglist.te[loglist.n_te] = IntToAlphabet(move.col);
    loglist.te[loglist.n_te++] += move.row;
    number++;
    /* クライアントにメッセージを送る */
    sprintf(buf, "%d %d\n", move.row, move.col);
    send(fd, buf, strlen(buf), 0);
    printf("● 1: 縦=%d 横=%d buf=%s\n", move.row, move.col, buf);
    game->showBoard();
    PassP = false;
}
/* 手を受信する */
/* クライアントからの入力待ち */
int  buflen = recv(fd, buf, strlen(buf)-1, 0);;
if ( buflen > 0) {
```

```
                        /* 文字列として扱うために NULL 文字を追加 */
                        buf[buflen] = '\0';
                        who = WHITE;
                        movelist.clear();
                        game->GenAllMoves(movelist, who);
                        if (movelist.size() == 0) {
                            /* 入力された文字列が「PS」なら OK*/
                            sscanf(buf, "%d %d", &move.row, &move.col);
                            if (move.row == 0 && move.col == 0) {
                                if (PassP) {
                                    loglist.te[loglist.n_te++] = "PS";
                                    number++;
                                    printf("○ C PS\n");
                                    game->showBoard();
                                    break;
                                } else {
                                    PassP = true;
                                    loglist.te[loglist.n_te++] = "PS";
                                    number++;
                                    printf("○ D PS buf=%s\n", buf);
                                    game->showBoard();
                                }
                            } else {
                                sscanf(buf, "%d %d", &move.row, &move.col);
                                game->add(move.row, move.col, who);
                                loglist.te[loglist.n_te] = IntToAlphabet(move.col);
                                loglist.te[loglist.n_te++] += move.row;
                                number++;
                                printf("○ 2: 縦=%d 横＝%d buf=%s\n", move.row, move.col, buf);
                                game->showBoard();
                                PassP = false;
                            }
                        }
                    }
                    /* クライアントとの接続を閉じる */
                    closesocket(fd);
                }
            }
        }
        /* ソケットを閉じる */
        closesocket(sock);
    } else {
```

```
/* クライアント側 */
/* 接続する */
ret = connect(sock, (const struct sockaddr *)&addr, sizeof(addr));

if (ret == 0) {
    char str[256];
    int len;

    /* 接続できたので、メッセージを受け取る */
    len = recv(sock, str, sizeof(str) - 1, 0);
    if (len > 0) {
        /* 文字列として扱いたいので NULL 文字をセット */
        str[len] = 0;
        /* 受信したメッセージを出力 */
        printf("%s", str);
        strcpy(str, "OK");
        /* 送信する */
        send(sock, str, strlen(str), 0);
    }
}
/* ソケットを閉じる */
closesocket(sock);

while (true) {
    /* ソケットを作成する */
    sock = socket(PF_INET, SOCK_STREAM, 0);
    /* 接続先の設定 */
    memset(&addr, 0, sizeof(addr));
    addr.sin_family = PF_INET;
    addr.sin_port = htons(50000);
    addr.sin_addr.s_addr = inet_addr("127.0.0.1");

    /* サーバーに接続する */
    ret = connect(sock, (const struct sockaddr *)&addr, sizeof(addr));

    if (ret == 0) {
        char str[256];
        /* 接続できたので、メッセージを受け取る */
        int len = recv(sock, str, sizeof(str) - 1, 0);
        if (len > 0) {
            /* 文字列として扱いたいので NULL 文字をセット */
            str[len] = 0;
            who = BLACK;
```

```
movelist.clear();
game->GenAllMoves(movelist, who);
if (movelist.size() == 0) {
    sscanf(str, "%d %d", &move.row, &move.col);
        if (move.row == 0 && move.col == 0) {
            if (PassP) {
                loglist.te[loglist.n_te++] = "PS";
                number++;
                /* ソケットを閉じる */
                printf("● E PS");
                game->showBoard();
                closesocket(sock);
                break;
            }
            PassP = true;
            loglist.te[loglist.n_te++] = "PS";
            number++;
            printf("● F PS str=$s\n", str);
            game->showBoard();
        }
    } else {
        sscanf(str, "%d %d", &move.row, &move.col);
        game->add(move.row, move.col, who);
        loglist.te[loglist.n_te] = IntToAlphabet(move.col);
        loglist.te[loglist.n_te++] += move.row;
        number++;
        printf("● 3: 縦=%d 横=%d str=%s\n", move.row, move.col, str);
        game->showBoard();
        PassP = false;
    }
}
/* サーバーにメッセージを送る */
who = WHITE;
movelist.clear();
game->GenAllMoves(movelist, who);
if (movelist.size() == 0) {
    if (PassP) {
        /* ホストにメッセージを送る */
        move.row = move.col = 0;
        sprintf(str, "%d %d\n", move.row, move.col);
        /* 送信する */
        send(sock, str, strlen(str), 0);
        printf("○ G PS   str=%s\n", str);
```

```
                        game->showBoard();
                        break;
                    } else {
                        PassP = true;
                        loglist.te[loglist.n_te++] = "PS";
                        number++;
                        /* ホストにメッセージを送る */
                        move.row = move.col = 0;
                    sprintf(str, "%d %d\n", move.row, move.col);
                    /* 送信する */
                    send(sock, str, strlen(str), 0);
                    printf("○ H PS str=%s\n", str);
                    game->showBoard();
                }
            } else {
                CPoint move = game->ComputerMove(who, UCT);
                game->add(move.row, move.col, who);
                loglist.te[loglist.n_te] = IntToAlphabet(move.col);
                loglist.te[loglist.n_te++] += move.row;
                number++;
                /* ホストにメッセージを送る */
                sprintf(str, "%d %d\n", move.row, move.col);
                /* 送信する */
                send(sock, str, strlen(str), 0);
                printf("○ 4: 縦=%d 横=%d str=%s\n", move.row, move.col, str);
                game->showBoard();
                PassP = false;
            }
        }
        /* ソケットを閉じる */
        closesocket(sock);
    }
}
who = BLACK;
int value = game->Calculate2(who);
if (value > 0) {
    string str = "●の ";
    string str2 = " 個の勝ちです。頑張ってね。";
    cout << str.c_str() << value << str2.c_str() << endl;
} else if (value < 0) {
    string str = "○の ";
    string str2 = " 個の勝ちです。強いですね。";
    cout << str.c_str() << -value << str2.c_str() << endl;
```

```
    } else {
        string str = "引き分けです。もう一度やりましょう！";
        cout << str.c_str() << endl;
    }

    /* Winsock の後処理 */
    WSACleanup();
    getchar();
    getchar();
    return 0;
}
```

インターネットでつながった二台のコンピュータでこのプログラムを動かします。但し、二ヶ所ある

```
/* 接続先の設定 */
memset(&addr, 0, sizeof(addr));
addr.sin_family = PF_INET;
addr.sin_port = htons(50000);
addr.sin_addr.s_addr = inet_addr("127.0.0.1");
```

の ”127.0.0.1” にはサーバーのアドレスをセットします。先手がサーバー側でまず起動し、後手のクライアント側をその後起動します。

```
#define uct_max 100
#define max_depth 12
#define MAXUCTLOOP 100000
#define THREAD_NUM 8
```

で、強さを加減します。 Windows フォームアプリケーションとして実現するには、コンピュータの手を生成する部分を DLL として作るのが最善みたいです。そうすれば、通常の参考書に書いてある上で述べたマルチスレッドの技法が使えます。更に、持ち時間が指定された時の対策もやってみるべきです。公開されているオープンソースのゲームのプログラムを読んでみるのもいいですが、多くの場合、Linux の C 言語で書かれたプログラムで、非常に複雑で理解するのが大変です。基本的で単純なプログラミングの勉強をし、単純なプログラムを色々作りながら、思いついたことを試行錯誤しながら、マルチスレッドや通信のプログラムが自分の頭で明確にイメージでき、理解できるよう、少しづつ本当に作りたいプログラムに近づけていくのが良いです。自分で苦労して見つけたことはその人の宝物になります。本当は囲碁のプログラムを作りたくてもいきなり難しいことをするのではなく、上でやったように、オセロのような単純なプログラムを作ってみて、経験を積んでから、囲碁のプログラムに進むのが良いです。プログラミングを始めた時は、なんでそんなことが出来るのか色々なことが不思議でしたけれど、その解説の載っている本を見つけ、プログラムを片っ端から読んでみて、不思議だと思うことをするためには、不思議と思うことをするようプログラムにコツコツと書いているし、人間がしていることを計算機がするようにプログラムにコツコツ書くしかないことを心底理解したときに、プログラムが自由に作れるようになりました。そのためには単純なプログラムを兎も角沢山読み、そして沢山作ってみます。

コンピュータゲームのプログラムを作りたければ、小谷善行編著岸本章宏・柴原一友・鈴木豪共著「ゲーム計算メカニズムー将棋・囲碁・オセロ・チェスのプログラムはどう動くー」コロナ社を読めば良いです。この本はゲームの技法をまとめて解説している多分唯一の本なので 2013 年冬に初めて真面目に読んでみました。プログラミングの初心者がこの本だけを読んでもコンピュータゲームのプログラムを実際に作るのは難しいと思います。「分かる」と言うことと「出来る」と言うことの間には大きな溝があります。インターネットでミニマックス法とかアルファベータ法を検索すれば、 Kalah （アフリカのゲームでオワリとも言います）を題材にミニマックス法とアルファベータ法、更に、ネガマックス法とネガスカウト法、置換表と MTD(f) 法、８パズルを題材に幅優先探索と反復深化、ヒューリスティック探索を Python のプログラムで解説したホームページを見つけることが出来ます。そこで、一か月ぐらいかけて Python を勉強し、上記の本とこのホームページを見比べながら精読すれば、上記の本の半分ぐらいは理解できるようになります。上記の本の１０章　AND/OR 木と証明数探索からは自力で読まなければなりませんが、演習問題にあるドミニアリング（これはクロスクラムとも言います）の証明数探索のプログラムを Python で作ります。ちなみに、私が 大昔、アルファベータ法を勉強し、BASIC で初めて作ったゲームのプログラムがクロスクラムのプログラムです。

まずクロスクラムのゲームのプログラムを作ります。

```
FIRST_PLAYER = 0
SECOND_PLAYER = 1

class Board:
    def __init__(self, b, n, m):
        self.board = b[:]
        self.width = n
        self.hight = m
    def __getitem__(self, x):
        return self.board[x]
    def copy(self):
        return Board(self.board, self.width, self.hight)
    def put_piece(self, turn, pos):
        if turn == FIRST_PLAYER:
            self.board[pos] = 1
            self.board[pos+1] = 1
        else:
            self.board[pos] = 2
            self.board[pos+self.width+1] = 2
    def regalmoveP(self, turn, pos):
        if turn == FIRST_PLAYER:
            if self.board[pos] == 0 and self.board[pos+1] == 0:
                return True
            else:
                return False
        else:
```

```
            if self.board[pos] == 0 and self.board[pos+self.width+1] == 0:
                return True
            else:
                return False
    def print_board(self):
        for y in range(self.hight):
            for x in range(self.width):
                print self.board[(self.width+1)*y+x],
            print
        print
    def get_candidate(self, turn):
        cand = []
        if turn == FIRST_PLAYER:
            for x in range(self.width-1):
                for y in range(self.hight):
                    if (self.regalmoveP(FIRST_PLAYER, (self.width+1)*y+x)):
                        cand.append((self.width+1)*y+x)
        else:
            for x in range(self.width):
                for y in range(self.hight-1):
                    if (self.regalmoveP(SECOND_PLAYER, (self.width+1)*y+x)):
                        cand.append((self.width+1)*y+x)
        return cand
    def gameoverP(self, turn):
        cand = self.get_candidate(turn)
        if len(cand) == 0:
            return True
        else:
            return False

import random, sys, re

def play(n, m):
    l = [0] * n + [1]
    ls = l * m
    ls += [1] * (n + 1)
    board = Board(ls, n, m)
    turn = FIRST_PLAYER
    while True:
        board.print_board()
        if turn == FIRST_PLAYER:
            print "FIRST_PLAYER:"
        else:
```

```
            print "SECOND_PLAYER:"
        if board.gameoverP(turn):
            print 'Game Over'
            break
        if turn == FIRST_PLAYER:
            while True:
                print "?=",
                s = raw_input()
                pat = re.compile('\W+')
                sl = pat.split(s)
                [sn, sm] = sl
                pos = int(sm)*(n+1)+int(sn)
                if board.regalmoveP(turn, pos):
                    board.put_piece(turn, pos)
                    break
        else:
            cand = board.get_candidate(turn)
            ind = random.randint(0, len(cand)-1)
            board.put_piece(turn, cand[ind])
        if turn == FIRST_PLAYER:
            turn = SECOND_PLAYER
        else:
            turn = FIRST_PLAYER

if __name__ == '__main__':
    play(4, 4)
```

次に単純な探索によるプログラムを作ります。複雑なプログラムの性能判定やデバッグに役立ちます。

```
import time
from crosscram2 import *

def simple_search(n, m):
    l = [0] * n + [1]
    ls = l * m
    ls += [1] * (n + 1)
    print "width=%d hight=%d" % (n, m)
    board = Board(ls, n, m)
    turn = FIRST_PLAYER
    value = move_first(board)
    if value:
       print "sente win"
    else:
```

```
        print "gote win"

def move_first(board):
    cand = board.get_candidate(FIRST_PLAYER)
    if len(cand) == 0:
        return False
    for x in cand:
        b = board.copy()
        b.put_piece(FIRST_PLAYER, x)
        value = move_second(b)
        if value:
            return True
    return False

def move_second(board):
    cand = board.get_candidate(SECOND_PLAYER)
    if len(cand) == 0:
       return True
    for x in cand:
        b = board.copy()
        b.put_piece(SECOND_PLAYER, x)
        value = move_first(b)
        if not value:
            return False
    return True

s = time.clock()
simple_search(5, 5)
e = time.clock()
print "%.3f" % (e - s)
```

本にはアルゴリズムが簡潔に書いてあります。

証明数を OR ノードの評価関数、反証数を AND ノードの評価関数であると考えると、次のような最良優先探索アルゴリズムが自然に定義できる。この最良優先探索を証明数探索 (proof number search) と呼ぶ。

1 ルートノードから開始する。
2 OR ノードでは、証明数が最小の子ノードを選択する。
3 AND ノードでは、反証数が最小の子ノードを選択する。
4 ステップ 2 と 3 を先端ノードに至るまで繰り返す。この先端ノードを最有力ノードと呼ぶ。
5 最有力ノードを 1 段階のみ展開する。
6 ステップ 2 から 5 で選択した探索経路を戻り、証明数・反証数を再計算する。
7 ステップ 1 から 6 をルートノードが解けるまで繰り返す。

この記述（アルゴリズム）を元に私が最初に作ったドミニアリング（これはクロスクラムとも言い

ます）の証明数探索のプログラムは次のようなものです。

```
import time
from crosscram2 import *

INFINITY = 1000000

class State:
    def __init__(self, p_num, d_num):
        self.p_num = p_num
        self.d_num = d_num

def addition(n, m):
    if n == INFINITY or m == INFINITY:
        return INFINITY
    else:
        return n + m

def proof_number_search(n, m):
    l = [0] * n + [1]
    ls = l * m
    ls += [1] * (n + 1)
    board = Board(ls, n, m)
    turn = FIRST_PLAYER
    print "width = %d hight = %d" % (n, m)
    while True:
        p_num, d_num = search_or(board)
        if p_num == 0 and d_num == INFINITY:
            print "sente win"
            break
        elif p_num == INFINITY and d_num == 0:
            print "gote win"
            break

def search_or(board):
    cand = board.get_candidate(FIRST_PLAYER)
    if len(cand) == 0:
        # terminal node
        table[tuple(board)] = State(INFINITY, 0)
        return INFINITY, 0
    key = tuple(board)
    if key not in table:
        # not expand node
```

```
        table[tuple(board)] = State(1, 1)
        return 1, 1
    else:
        d = table[key]
        if d.p_num == INFINITY and d.d_num == 0:
            return INFINITY, 0
        elif d.p_num == 0 and d.d_num == INFINITY:
            return 0, INFINITY

        # expand node
        flag = True
        for x in cand:
            b = board.copy()
            b.put_piece(FIRST_PLAYER, x)
            k = tuple(b)
            if k not in table:
                flag = False
                break
        if flag:
            # interior node
            while True:
                old_p_num = table[key].p_num
                old_d_num = table[key].d_num
                min_p_num = INFINITY
                min_x = None
                true_flag = False
                false_flag = True
                prev_p_num = None
                prev_d_num = None
                p = []
                d = []
                for x in cand:
                    b = board.copy()
                    b.put_piece(FIRST_PLAYER, x)
                    k = tuple(b)
                    c = table[k]
                    p.append(c.p_num)
                    d.append(c.d_num)
                    if c.p_num == 0 and c.d_num == INFINITY:
                        true_flag = True
                    if c.p_num != INFINITY or c.d_num != 0:
                        false_flag = False
                    if c.p_num < min_p_num:
```

```
            min_p_num = c.p_num
            min_x = x
            prev_p_num = c.p_num
            prev_d_num = c.d_num
        elif c.p_num == min_p_num and min_p_num == INFINITY:
            min_x = x
            prev_p_num = c.p_num
            prev_d_num = c.d_num
    if true_flag:
        table[tuple(board)] = State(0, INFINITY)
        return 0, INFINITY
    if false_flag:
        table[tuple(board)] = State(INFINITY, 0)
        return INFINITY, 0
    if prev_p_num == INFINITY and prev_d_num == 0:
        table[tuple(board)] = State(INFINITY, 0)
        return INFINITY, 0
    b = board.copy()
    b.put_piece(FIRST_PLAYER, min_x)
    new_p_num, new_d_num = search_and(b)
    # modify p_num and d_num
    p = []
    d = []
    for x in cand:
        b = board.copy()
        b.put_piece(FIRST_PLAYER, x)
        k = tuple(b)
        c = table[k]
        p.append(c.p_num)
        d.append(c.d_num)
    # p_num = min(pnum(n1), pnum(n2), ... , pnum(nk))
    p_num = INFINITY
    for n in p:
        if n < p_num:
            p_num = n
    # d_num = dnum(n1)+dnum(n2)+ ... +dnum(nk)
    d_num = 0
    for n in d:
        d_num = addition(d_num, n)
    if p_num == INFINITY and d_num == 0:
        table[tuple(board)] = State(p_num, d_num)
        return p_num, d_num
    elif p_num == 0 and d_num == INFINITY:
```

```
                    table[tuple(board)] = State(p_num, d_num)
                    return p_num, d_num
                else:
                    if old_p_num != p_num or old_d_num != d_num:
                        table[tuple(board)] = State(p_num, d_num)
                        return p_num, d_num
                    else:
                        continue
        else:
            # leaf node
            p = []
            d = []
            for x in cand:
                b = board.copy()
                b.put_piece(FIRST_PLAYER, x)
                k = tuple(b)
                if k in table:
                    c = table[k]
                    p.append(c.p_num)
                    d.append(c.d_num)
                else:
                    p.append(1)
                    d.append(1)
                    table[tuple(b)] = State(1, 1)
            # p_num = min(pnum(n1), pnum(n2), ... , pnum(nk))
            p_num = INFINITY
            for n in p:
                if n < p_num:
                    p_num = n
            # d_num = dnum(n1)+dnum(n2)+ ... +dnum(nk)
            d_num = 0
            for n in d:
                d_num = addition(d_num, n)
            table[tuple(board)] = State(p_num, d_num)
            return p_num, d_num

def search_and(board):
    cand = board.get_candidate(SECOND_PLAYER)
    if len(cand) == 0:
        # terminal node
        table[tuple(board)] = State(0, INFINITY)
        return 0, INFINITY
    key = tuple(board)
```

```
if key not in table:
    # not expand node
    table[tuple(board)] = State(1, 1)
    return 1, 1
else:
    # expand node
    d = table[key]
    if d.p_num == 0 and d.d_num == INFINITY:
        return 0, INFINITY
    elif d.p_num == INFINITY and d.d_num == 0:
        return INFINITY, 0
    flag = True
    for x in cand:
        b = board.copy()
        b.put_piece(SECOND_PLAYER, x)
        k = tuple(b)
        if k not in table:
            flag = False
            break
    if flag:
        # interior node
        while True:
            old_p_num = table[key].p_num
            old_d_num = table[key].d_num
            min_d_num = INFINITY
            min_x = None
            prev_p_num = None
            prev_d_num = None
            false_flag = False
            true_flag = True
            p = []
            d = []
            for x in cand:
                b = board.copy()
                b.put_piece(SECOND_PLAYER, x)
                k = tuple(b)
                c = table[k]
                p.append(c.p_num)
                d.append(c.d_num)
                if c.p_num != 0 and c.d_num != INFINITY:
                    true_flag = False
                if c.p_num == INFINITY or c.d_num == 0:
                    false_flag = True
```

```
    if c.d_num < min_d_num:
        min_d_num = c.d_num
        min_x = x
        prev_p_num = c.p_num
        prev_d_num = c.d_num
    elif c.d_num == min_d_num and min_d_num == INFINITY:
        min_x = x
        prev_p_num = c.p_num
        prev_d_num = c.d_num
if true_flag:
    table[tuple(board)] = State(0, INFINITY)
    return 0, INFINITY
if false_flag:
    table[tuple(board)] = State(INFINITY, 0)
    return INFINITY, 0
if prev_p_num == 0 and prev_d_num == INFINITY:
    table[tuple(board)] = State(0, INFINITY)
    return 0, INFINITY
b = board.copy()
b.put_piece(SECOND_PLAYER, min_x)
new_p_num, new_d_num = search_or(b)
# modify p_num and d_num
p = []
d = []
for x in cand:
    b = board.copy()
    b.put_piece(SECOND_PLAYER, x)
    k = tuple(b)
    c = table[k]
    p.append(c.p_num)
    d.append(c.d_num)
# p_num = pnum(n1)+pnum(n2)+ ... +pnum(nk))
p_num = 0
for n in p:
    p_num = addition(p_num, n)
# d_num = dnum(n1)+dnum(n2)+ ... +dnum(nk)
d_num = INFINITY
for n in d:
    if n < d_num:
        d_num = n
if p_num == 0 and d_num == INFINITY:
    table[tuple(board)] = State(p_num, d_num)
    return 0, INFINITY
```

```
                elif p_num == INFINITY and d_num == 0:
                    table[tuple(board)] = State(p_num, d_num)
                    return INFINITY, 0
                else:
                    if old_p_num != p_num or old_d_num != d_num:
                        table[tuple(board)] = State(p_num, d_num)
                        return p_num, d_num
                    else:
                        continue
        else:
            # leaf node
            p = []
            d = []
            for x in cand:
                b = board.copy()
                b.put_piece(SECOND_PLAYER, x)
                k = tuple(b)
                if k in table:
                    c = table[k]
                    p.append(c.p_num)
                    d.append(c.d_num)
                else:
                    p.append(1)
                    d.append(1)
                    table[tuple(b)] = State(1, 1)
            # p_num = pnum(n1)+pnum(n2)+ ... +pnum(nk))
            p_num = 0
            for n in p:
                p_num = addition(p_num, n)
            # d_num = min(dnum(n1), dnum(n2), ... , dnum(nk))
            d_num = INFINITY
            for n in d:
                if n < d_num:
                    d_num = n
            table[tuple(board)] = State(p_num, d_num)
            return p_num, d_num

s = time.clock()
table = {}
proof_number_search(5, 5)
e = time.clock()
print "%.3f" % (e - s)
```

間違っていないと思いますが作り方にまずいところがあり、このプログラムでは5×5の盤で、私のノートパソコンでは 87.247 秒掛かりますが、単純な探索プログラムでは 51.464 秒でした。ちなみに結果は後手の勝ちです。

ひとまずこれで良いことにして、次を読んでいきます。証明数を用いた反復深化法やトランスポジションテーブルの利用などの説明があり、次々下手くそなプログラムを作りながら本を読んでいくと 111 ページから背尾のアルゴリズムの疑似コードが載っています。これをクロスクラムをテーマに Python でプログラミングしてみると次のようになります。

```
# coding: shift_jis

import time

FIRST_PLAYER = 0
SECOND_PLAYER = 1

class Board:
    def __init__(self, b, n, m, threshold):
        self.board = b[:]
        self.width = n
        self.hight = m
        self.threshold = threshold
    def __getitem__(self, x):
        return self.board[x]
    def copy(self):
        return Board(self.board, self.width, self.hight, self.threshold)
    def put_piece(self, turn, pos):
        if turn == FIRST_PLAYER:
            self.board[pos] = 1
            self.board[pos+1] = 1
        else:
            self.board[pos] = 2
            self.board[pos+self.width+1] = 2
    def regalmoveP(self, turn, pos):
        if turn == FIRST_PLAYER:
            if self.board[pos] == 0 and self.board[pos+1] == 0:
                return True
            else:
                return False
        else:
            if self.board[pos] == 0 and self.board[pos+self.width+1] == 0:
                return True
            else:
                return False
```

```
    def print_board(self):
        for y in range(self.hight):
            for x in range(self.width):
                print self.board[(self.width+1)*y+x],
            print
        print "threshold=%d" % self.threshold
    def get_candidate(self, turn):
        cand = []
        if turn == FIRST_PLAYER:
            for x in range(self.width-1):
                for y in range(self.hight):
                    if (self.regalmoveP(FIRST_PLAYER, (self.width+1)*y+x)):
                        cand.append((self.width+1)*y+x)
        else:
            for x in range(self.width):
                for y in range(self.hight-1):
                    if (self.regalmoveP(SECOND_PLAYER, (self.width+1)*y+x)):
                        cand.append((self.width+1)*y+x)
        return cand
    def gameoverP(self, turn):
        cand = self.get_candidate(turn)
        if len(cand) == 0:
            return True
        else:
            return False

INFINITY = 1000000

def addition(n, m):
    if n == INFINITY or m == INFINITY:
        return INFINITY
    else:
        return n + m

# 背尾のアルゴリズム
def seo(n, m):
    l = [0] * n + [1]
    ls = l * m
    ls += [1] * (n + 1)
    print "width = %d hight = %d" % (n, m)
    root = Board(ls, n, m, 2)
    turn = FIRST_PLAYER
    while True:
```

```
        # ルートノードで反復深化を行う
        value = MID(root, turn)
        if value == True:
            print "sente win"
            break
        elif value == False:
            print "gote win"
            break
        # 解けないときには閾値を増やす
        root.threshold += 1

# 多重反復深化を行う関数
def MID(node, turn):
    p_num = TTLookUp(node)
    if p_num == 0 or p_num == INFINITY:
        if p_num == 0:
            return True
        else:
            return False
    if p_num >= node.threshold:
        return None
    children = Generate(node, turn)
    if len(children) == 0:
        if turn == FIRST_PLAYER:
            TTSave(node, INFINITY)
            return False
        else:
            TTSave(node, 0)
            return True
    if turn == FIRST_PLAYER: # node が OR ノード
        for child_node in children:
            child_node.threshold = node.threshold
            if MID(child_node, SECOND_PLAYER):
                TTSave(node, 0)
                return True
        # 証明数の再計算
        pn_min = MinPn(node, children)
        TTSave(node, pn_min)
        if pn_min == INFINITY:
            return False
        return None
    else: # node が AND ノード
        while node.threshold > SumPn(node, children):
```

```
            child_node = FindUnprovenChild(node, children)
            if child_node == None:
                break
            child_node.threshold += 1
            MID(child_node, FIRST_PLAYER)
        pn_sum = SumPn(node, children)
        TTSave(node, pn_sum)
        if pn_sum == 0 or pn_sum == INFINITY:
            if pn_sum == 0:
                return True
            else:
                return False
        return None

# 子ノードを返す
def Generate(node, turn):
    cand = node.get_candidate(turn)
    children = []
    p = []
    i = 0
    for pos in cand:
        b = node.copy()
        b.put_piece(turn, pos)
        p.append(TTLookUp(b))
        children.append(b)
        i += 1
        k = i-1
        while k > 0:
            if p[k] < p[k-1]:
                temp = p[k]
                p[k] = p[k-1]
                p[k-1] = temp
                junk = children[k]
                children[k] = children[k-1]
                children[k-1] = junk
                k -= 1
            else:
                break
    return children

# 子ノードで最小の証明数を求める
def MinPn(node, children):
    pn_min = INFINITY
```

```
    for child_node in children:
        pn = TTLookUp(child_node)
        pn_min = min(pn_min, pn)
    return pn_min

# 子ノードの証明数の和を計算
def SumPn(node, children):
    pn_sum = 0
    for child_node in children:
        pn = TTLookUp(child_node)
        pn_sum = addition(pn_sum, pn)
    return pn_sum

# 証明されていない子ノードを返す
def FindUnprovenChild(node, children):
    for child_node in children:
        p_num = TTLookUp(child_node)
        if p_num != 0:
            return child_node
    return None

# トランスポジションテーブルの参照
def TTLookUp(node):
    key = tuple(node.board)
    if key in table:
        p_num = table[key]
    else:
        p_num = 1
    return p_num

# トランスポジションテーブルへの保存
def TTSave(node, p_num):
    key = tuple(node.board)
    if key in table:
        if p_num == 0 or p_num > table[key]:
            # 証明されたノードか
            # 以前より大きな証明数を保存
            table[key] = p_num
    else:
        table[key] = p_num


if __name__ == '__main__':
```

```
    s = time.clock()
    table = {}
    seo(6,6)
    e = time.clock()
    print "%.3f" % (e - s)
```

となります。5 × 5 の盤で後手勝ち 5.786 秒、6 × 6 の盤で先手勝ち 714.381 秒です。このプログラムを見ると以前作った証明数探索のプログラムは

```
# coding: shift_jis

import time
from crosscram2 import *

INFINITY = 1000000

class State:
    def __init__(self, p_num, d_num):
        self.p_num = p_num
        self.d_num = d_num

def addition(n, m):
    if n == INFINITY or m == INFINITY:
        return INFINITY
    else:
        return n + m

def proof_number_search(n, m):
    l = [0] * n + [1]
    ls = l * m
    ls += [1] * (n + 1)
    board = Board(ls, n, m)
    turn = FIRST_PLAYER
    print "width=%d hight=%d" % (n, m)
    while True:
        value = search_or(board)
        if value == True:
            print "sente win"
            break
        elif value == False:
            print "gote win"
            break

def search_or(board):
```

```
    p_num, d_num = TTLookUp(board)
    if p_num == 0:
        return True
    if p_num == INFINITY:
        return False
    children = Generate_OR(board, FIRST_PLAYER)
    if len(children) == 0:
        # terminal node
        TTSave(board, INFINITY, 0)
        return False
    for child in children:
        if search_and(child) == True:
            TTSave(board, 0, INFINITY)
            return True
    # modify p_num and d_num
    pn_min = MinPn(children)
    dn_sum = SumDn(children)
    TTSave(board, pn_min, dn_sum)
    if pn_min == INFINITY:
        return False
    return None

def search_and(board):
    p_num, d_num = TTLookUp(board)
    if p_num == 0:
        return True
    if p_num == INFINITY:
        return False
    children = Generate_AND(board, SECOND_PLAYER)
    if len(children) == 0:
        # terminal node
        TTSave(board, 0, INFINITY)
        return True
    for child in children:
        if search_or(child) == False:
            TTSave(board, INFINITY, 0)
            return False
    # modify p_num and d_num
    pn_sum = SumPn(children)
    dn_min = MinDn(children)
    TTSave(board, pn_sum, dn_min)
    if pn_sum == 0:
        return True
```

```
    elif pn_sum == INFINITY:
        return False
    return None

# 子ノードを返す
def Generate_OR(node, turn):
    cand = node.get_candidate(turn)
    children = []
    p = []
    i = 0
    for pos in cand:
        b = node.copy()
        b.put_piece(turn, pos)
        p_num, d_num = TTLookUp(b)
        p.append(p_num)
        children.append(b)
        i += 1
        k = i-1
        while k > 0:
            if p[k] < p[k-1]:
                temp = p[k]
                p[k] = p[k-1]
                p[k-1] = temp
                junk = children[k]
                children[k] = children[k-1]
                children[k-1] = junk
                k -= 1
            else:
                break
    return children

def Generate_AND(node, turn):
    cand = node.get_candidate(turn)
    children = []
    d = []
    i = 0
    for pos in cand:
        b = node.copy()
        b.put_piece(turn, pos)
        p_num, d_num = TTLookUp(b)
        d.append(d_num)
        children.append(b)
        i += 1
```

```
        k = i-1
        while k > 0:
            if d[k] < d[k-1]:
                temp = d[k]
                d[k] = d[k-1]
                d[k-1] = temp
                junk = children[k]
                children[k] = children[k-1]
                children[k-1] = junk
                k -= 1
            else:
                break
    return children

# 子ノードで最小の証明数を求める
def MinPn(children):
    pn_min = INFINITY
    for child_node in children:
        pn, d_num = TTLookUp(child_node)
        pn_min = min(pn_min, pn)
    return pn_min

# 子ノードの証明数の和を計算
def SumPn(children):
    pn_sum = 0
    for child_node in children:
        pn, d_num = TTLookUp(child_node)
        pn_sum = addition(pn_sum, pn)
    return pn_sum

# 子ノードで最小の証明数を求める
def MinDn(children):
    dn_min = INFINITY
    for child_node in children:
        p_num, dn = TTLookUp(child_node)
        dn_min = min(dn_min, dn)
    return dn_min

# 子ノードの証明数の和を計算
def SumDn(children):
    dn_sum = 0
    for child_node in children:
        p_num, dn = TTLookUp(child_node)
```

```
        dn_sum = addition(dn_sum, dn)
    return dn_sum

def TTLookUp(node):
    key = tuple(node)
    if key in table:
       c = table[key]
       p_num = c.p_num
       d_num = c.d_num
    else:
       p_num = 1
       d_num = 1
    return p_num, d_num

def TTSave(node, p_num, d_num):
    key = tuple(node)
    if key in table:
       c = table[key]
       if p_num == 0 or p_num > c.p_num or d_num == 0 or d_num > c.d_num:
           table[key] = State(p_num, d_num)
           table[key].p_num = p_num
       if d_num == 0 or d_num > table[key].d_num:
           table[key].d_num = d_num
    else:
        table[key] = State(p_num, d_num)

s = time.clock()
table = {}
proof_number_search(5, 5)
e = time.clock()
print "%.3f" % (e - s)
```

のように作るべきであったことが分かります。今度は 5×5 の盤で 19.445 秒で答えが出ます。

次の df-pn アルゴリズムの Python によるプログラムは

```
# coding: shift_jis

import time

FIRST_PLAYER = 0
SECOND_PLAYER = 1

class Board:
    def __init__(self, b, n, m, th_p_num, th_d_num):
```

```
    self.board = b[:]
    self.width = n
    self.hight = m
    self.th_p_num = th_p_num
    self.th_d_num = th_d_num
def __getitem__(self, x):
    return self.board[x]
def copy(self):
    return Board(self.board, self.width, self.hight, \
                 self.th_p_num, self.th_d_num)
def put_piece(self, turn, pos):
    if turn == FIRST_PLAYER:
        self.board[pos] = 1
        self.board[pos+1] = 1
    else:
        self.board[pos] = 2
        self.board[pos+self.width+1] = 2
def regalmoveP(self, turn, pos):
    if turn == FIRST_PLAYER:
        if self.board[pos] == 0 and self.board[pos+1] == 0:
            return True
        else:
            return False
    else:
        if self.board[pos] == 0 and self.board[pos+self.width+1] == 0:
            return True
        else:
            return False
def print_board(self):
    for y in range(self.hight):
        for x in range(self.width):
            print self.board[(self.width+1)*y+x],
        print
    print "th_p_num=%d th_d_num=%d" % (self.th_p_num, self.th_d_num)
def get_candidate(self, turn):
    cand = []
    if turn == FIRST_PLAYER:
        for x in range(self.width-1):
            for y in range(self.hight):
                if (self.regalmoveP(FIRST_PLAYER, (self.width+1)*y+x)):
                    cand.append((self.width+1)*y+x)
    else:
        for x in range(self.width):
```

```
                for y in range(self.hight-1):
                    if (self.regalmoveP(SECOND_PLAYER, (self.width+1)*y+x)):
                        cand.append((self.width+1)*y+x)
        return cand
    def gameoverP(self, turn):
        cand = self.get_candidate(turn)
        if len(cand) == 0:
            return True
        else:
            return False

INFINITY = 1000000

class State:
    def __init__(self, p_num, d_num):
        self.p_num = p_num
        self.d_num = d_num

def addition(n, m):
    if n == INFINITY or m == INFINITY:
        return INFINITY
    else:
        return n + m

def df_pn(n, m):
    l = [0] * n + [1]
    ls = l * m
    ls += [1] * (n + 1)
    root = Board(ls, n, m, INFINITY-1, INFINITY-1)
    print "width=%d hight=%d" % (n, m)
    p_num, d_num = MID_OR(root)
    if d_num == INFINITY:
        print "sente win"
    elif p_num == INFINITY:
        print "gote win"
    else:
        print "unkown"

def MID_OR(node):
    p_num, d_num = TTLookUp(node)
    if node.th_p_num <= p_num or node.th_d_num <= d_num:
        # 閾値を超えたとき
        node.th_p_num = p_num
```

```
        node.th_d_num = d_num
        return p_num, d_num
    children = Generate(node, FIRST_PLAYER)
    if len(children) == 0:
        # terminal node
        node.th_p_num = INFINITY
        node.th_d_num = 0
        TTSave(node, INFINITY, 0)
        return INFINITY, 0
    while node.th_p_num > MinPn(children) and \
          node.th_d_num > SumDn(children):
        child, dnc, pn2 = SelectChild_OR(children)
        # 閾値の設定
        child.th_p_num = min(node.th_p_num, pn2+1)
        child.th_d_num = node.th_d_num + dnc - SumDn(children)
        MID_AND(child)
    # 探索結果を保存
    p_num = MinPn(children)
    d_num = SumDn(children)
    TTSave(node, p_num, d_num)
    return p_num, d_num

def MID_AND(node):
    p_num, d_num = TTLookUp(node)
    if node.th_d_num <= d_num or node.th_p_num <= p_num:
        # 閾値を超えたとき
        node.th_p_num = p_num
        node.th_d_num = d_num
        return p_num, d_num
    children = Generate(node, SECOND_PLAYER)
    if len(children) == 0:
        # terminal node
        node.th_p_num = 0
        node.th_d_num = INFINITY
        TTSave(node, 0, INFINITY)
        return 0, INFINITY
    while node.th_d_num > MinDn(children) and node.th_p_num > SumPn(children):
        child, pnc, dn2 = SelectChild_AND(children)
        # 閾値の設定
        child.th_d_num = min(node.th_d_num, dn2+1)
        child.th_p_num = node.th_p_num + pnc - SumPn(children)
        MID_OR(child)
    # 探索結果を保存
```

```
    d_num = MinDn(children)
    p_num = SumPn(children)
    TTSave(node, p_num, d_num)
    return p_num, d_num

# 子ノードを返す
def Generate(node, turn):
    cand = node.get_candidate(turn)
    children = []
    for pos in cand:
        b = node.copy()
        b.put_piece(turn, pos)
        children.append(b)
    return children

# 最有力な子ノードを見つける
def SelectChild_OR(children):
    dc = pc = INFINITY
    p2 = INFINITY
    for child in children:
        p_num, d_num = TTLookUp(child)
        # dc に最小の d_num を入れ
        # d2 に二番目に小さな p_num を入れる
        if p_num < pc:
            n_best = child
            p2 = pc
            pc = p_num
            dc = d_num
        elif p_num < p2:
            p2 = p_num
    return n_best, dc, p2

# 最有力な子ノードを見つける
def SelectChild_AND(children):
    dc = pc = d2 = INFINITY
    for child in children:
        p_num, d_num = TTLookUp(child)
        # pc に最小の p_num を入れ
        # p2 に二番目に小さな p_num を入れる
        if d_num < dc:
            n_best = child
            d2 = dc
            pc = p_num
```

```
            dc = d_num
        elif d_num < d2:
            d2 = d_num
    return n_best, pc, d2

# 子ノードで最小の証明数を求める
def MinPn(children):
    pn_min = INFINITY
    for child_node in children:
        pn, d_num = TTLookUp(child_node)
        pn_min = min(pn_min, pn)
    return pn_min

# 子ノードの証明数の和を計算
def SumPn(children):
    pn_sum = 0
    for child_node in children:
        pn, d_num = TTLookUp(child_node)
        pn_sum = addition(pn_sum, pn)
    return pn_sum

# 子ノードで最小の証明数を求める
def MinDn(children):
    dn_min = INFINITY
    for child_node in children:
        p_num, dn = TTLookUp(child_node)
        dn_min = min(dn_min, dn)
    return dn_min

# 子ノードの証明数の和を計算
def SumDn(children):
    dn_sum = 0
    for child_node in children:
        p_num, dn = TTLookUp(child_node)
        dn_sum = addition(dn_sum, dn)
    return dn_sum

def TTLookUp(node):
    key = tuple(node)
    if key in table:
       c = table[key]
       p_num = c.p_num
       d_num = c.d_num
```

```
    else:
       p_num = 1
       d_num = 1
    return p_num, d_num

def TTSave(node, p_num, d_num):
    key = tuple(node)
    if key in table:
       table[key] = State(p_num, d_num)
    else:
        table[key] = State(p_num, d_num)

s = time.clock()
table = {}
df_pn(4, 4)
e = time.clock()
print "%.3f" % (e - s)
```

で良いと思いますが、今度は４×４の盤で 0.351 秒、５×５の盤で 112.822 秒で答えが出ますが、６×６の盤ではメモリを使い切って停止します。この問題の対策を知るためには松原仁編著「アマ４段を超える　コンピュータ将棋の進歩４」共立出版の５章「df-pn アルゴリズムと詰将棋を解くプログラムへの応用」も読んでみる必要があります。df-pn アルゴリズムの欠点とその対策が書いてあります。

他人が書いたプログラムを読んで理解するのは比較的簡単ですが、すぐ忘れます。実際に自分でゼロから（証明数探索を初めて勉強して）、覚えたての Python でプロトタイプを作るのは、C++ で本格的なプログラムを作るよりはるかに簡単ですが、それでも大変で私が作ると数日掛かりました。この様に「分かる」ことに満足せず、「出来る」ことを追及する必要があります。「分かる」ことと「出来る」事には大きな違いがあります。この様に、単純なモデルで出てくるアルゴリズムを一つ一つプログラムに変換しながら時間をかけて悪戦苦闘しながら勉強していくと難解な理論を解説した本でも理解できるようになります。

これらの理論を勉強しても強いゲームのプログラムは素人ではすぐには作れません。強いゲームのプログラムを作りたければ、まずは、沢山関連する書籍が出版されているコンピュータ将棋の勉強をしてみるのが良いと思います。今は絶版になって、古本屋で探すか図書館で探さないといけませんが、池康弘著「コンピュータ将棋のアルゴリズム」工学社に添付されているプログラムリストを読んで見ます。この本に添付されているプログラムリストは貴重です。しかし、この本も将棋の高段者でなければ途中でプログラムリストの意味が理解できなくなります。更に読み進むためには、小谷善行・吉川竹四朗・柿本義一・森田和郎共著「コンピュータ将棋　あなたも挑戦してみませんか」サイエンス社から初めて、松原仁編著「コンピュータ将棋の進歩」のシリーズ全６冊を読む必要があります。コンピュータ将棋の分野で活躍した方々（コンピュータ将棋を牽引し一時代を築いた森田和郎さんは去年お亡くなりになりました）の貴重な解説を読むことが出来ます。これらの本（その他にもコンピュータ将棋関連の本が何冊かあります）は発売と同時に購入し持っていましたが、日々の仕事・生活に追われ読んでいませんでした。年金生活になった今、暇になったので、埃を被っていた本たちを取り出し読んでみると面白いです。このように、コンピュータ将棋は

沢山の情報がありますが、一つ一つ読んで理解していくのは結構大変です。更に重要なことは、主として日本人が競争している将棋のプログラムの開発競争はもうすぐ終わります。2013 年の日本数学会の講演で、松原仁さんは次の目標は「接待将棋」だと言っていましたがどうでしょう。

コンピュータ囲碁は吉川竹四朗著「コンピュータ囲碁　GREAT - プログラムの作り方とネット対局の実際」エスアイビーアクセス（2001)、清慎一、山下宏、佐々木宣介著「コンピュータ囲碁の入門」共立出版（2005)、及び松原仁編、美添一樹・山下宏著「コンピュータ囲碁　モンテカルロ法の理論と実践」共立出版（2012）を読むと良いです。`Minorization-Maximization` アルゴリズムでは、各々のプレイヤーはそれぞれの着手の特徴を表し、チームは着手（着手の特徴の組み合わせ）を表し、棋譜においてプロ棋士によって打たれた手を勝者、他の候補手を敗者のゲームと考えてそれぞれの着手の特徴の確率を計算します。コンピュータ将棋は評価関数を機械学習でより良くしようとし、コンピュータ囲碁は着手の確率分布を機械学習でより良くしようとしていると思います。

一般的な機械学習の勉強には、無料で手に入るものとしては、Allen B. Downey 著「ThinkBayes」O'RELLY (この本の pdf を Green Tea Press から無料でゲットできます。Python のプログラムによる解説も付いていて理解しやすいお勧めの本です）や Trevor Hastie, Robert Tibshirani, Jerome Friedman 著「The Elements of Statistical Learning Data Mining, Inference, and Prediction」Springer （この本も `http://www-stat.stanford.edu/~tibs/` から無料で pdf がゲットできます）があります。更に、有名な Christopher.M.Bishop 著「Pattern Recognition and Machine Learning」（訳本「パターン認識と機械学習上・下」丸善+光成滋生著「パターン認識と機械学習の学習」暗黒通信団）や Richard S.Sutton and Andrew G. Barto 著「Reinforcement Learning An Introduction」（訳本「強化学習」森北出版）もプログラミングを勉強している人の常識として読むべきですが、残念ながら、これらの本もまだ埃を被って本箱で眠っています。

お金と時間が湯水のように必要で、お勧めはしませんが、なんの勉強でもそうだと思いますが、プログラミングはこのようにして技術を習得するものです。世の中を変えるような素晴らしいプログラムを作らなくても、上のような簡単なプログラムでも自力で作ることが出来ると充実感があります。ものを作るのは楽しいです。小中高の教員になる皆さんもどうせ趣味でプログラミングをしているので、何でも良いから自分で作りたいと思ったものをひたすら作れば良いです。少なくてもコンピュータは何を作っても笑ったりしません。文法的に間違っていれば、間違いを指摘してくれます。ただ、論理的な間違いは指摘してくれませんから、意図した通りに動かなければ、自分で間違いを見つけ修正する必要があります。自分の至らなさを清く受け入れることが必要です。間違っているところを探し、修正し、また失敗し、間違っているところを探し、修正し、また失敗し、を繰り返しているとそのうち正しく動くプログラムが出来上がります。ところがこれが厄介で多くの人がそれを我慢できずに挫折します。プログラミングで食べていける人は全人類の５０人に１人だそうです。大学・大学院で専門的な教育を受けたのは純粋数学（代数的位相幾何学・ホモトピー論の専攻）で、琉球大学理工学部の幾何学の助手時代に、数学科の学生さんたちと一緒に FORTRAN の自習書を読んで初めてコンピュータと付き合いだしてからずっと、プログラミングは暇を見つけて独学で学んできたのであって、コンピュータの専門教育を一度も受けたことのない私の講義を漠然と聞いて、プログラムをコピーして動かしてみてもプログラムが作れるようには決してなりません。Ruby を作った松本さんによると、彼の知っている有名なプログラマは全員、独学でコンピュータを学んだそうです。

昔、ネットワークについて話を高知大学にネットワークを導入する業者に聞いたり、Java によるネットワークプログラミングなどを本を読んで勉強したときは、いくらやっても全然頭に勉強したことが残りませんでした。そのときには、私の環境では物理的に複数のコンピュータをネット

ワークに繋げるわけでもなく、大学のコンピュータでホームページを作っても極端に制限された使い方しかできなかったので、ネットワークの勉強は私には本当には必要なかったからだと思います。ネットワークの仕組みが本質的に理解できていなかったので、騙されていても騙されていることに全然気づきませんでした。世間には色々な人達がいて、その人達なりの色々な事情が裏にあって、色々な人達が絡んで来た時に世間知らずだと自分の置かれている状況が良く呑み込めず失敗します。知らないということは悲しいことです。人生は失敗の連続です。いっぱい失敗して少しづつ賢くなります。最近、私のパソコンがマルチスレッドに対応しだし（CPU が i3 や i5 や i7 になった）、家のネットワーク環境にルータが導入され、複数のパソコンを有線や無線でインターネットに繋げるように知らない間になっていて（pikara がつながるようにしてくれていた）、やっとマルチスレッドやネットワークのプログラミングを実際に実験しながら勉強する物理的な勉強の環境が揃ってきました。金高堂でたまたま手に取った「C 言語逆引きハンドブック」という本にマルチスレッドやネットワークのプログラミングの基本が書いてありました。コンピュータの勉強は、唯本を読むだけの抽象的な勉強ではなく、「実際にプログラムを打ち込み、動かし、失敗し、何処が間違いか調べ、修正し、実際にプログラムを打ち込み、動かし、失敗し、何処が間違いか調べ、修正し、… 」の試行錯誤を繰り返しながら、集中的に時間を掛けて勉強する必要があります。そうすれば、次に何を勉強すればよいか自然に分かります。これは外国語の勉強と同じです。川島永嗣著「本当に「英語を話したい」キミへ」世界文化社が面白い本です。思い付きの教育法や高額の報酬を得て宣伝している誰かのような勉強法ではなく、実際に自分がやってヨーロッパの数か国語が話せるようになった勉強法が丁寧に書かれています。ともかく図書館や本屋にこまめに行って、目についた本を片っ端から読んでください。まとまった知識を得るには本が一番いいです。そのうえで、足りない情報をインターネットで探します。基本的な知識がないとインターネットで探しようがありません。

上のプログラムを理解するには、C++ のオブジェクト指向の勉強やポインタやポインタを使ったリストや木などのデータ構造の勉強が必要です。C++ の勉強をしてみたいと思ったら、 C++言語の開発者である Bjarne Stroustrup の「プログラミング言語 C++」や BJARNE STROUSTRUP 著「ストラウストラップのプログラミング入門　C++ によるプログラミングの原則と実践」　SE 2011
があります。これらが難しければ、翻訳は多分出ていませんが、「C++ Primer Plus」が英語ですが丁寧に説明されていて良いと思います。また、私は NIKLAUS WIRTH 著「Algorithms+Data Structures=Programs」 でアルゴリズムとデータ構造とプログラミングの考え方・作り方を学びましたが、今は絶版で、現在の推薦すべき本は知りません。自分で探してください。マルチスレッドの勉強には、「マルチコア CPU のための並列プログラミング　並列処理＆マルチスレッド入門」秀和システムを読んでから、「並行コンピューティング技法　実践マルチコア/マルチスレッドプログラミング」オライリー・ジャパンを読めばいいです。ネットワークのプログラミングの基本は、林晃著「C 言語逆引きハンドブック」　C&R 研究所　に載っている情報をもとに色々試行錯誤すれば良いです。

このテキストでは講義時間の関係もあり、昔の高等学校の数学の教科書に載っていたプログラムを中心に解説したので、C の基本的な概念がいっぱい抜けています。C の勉強は昔私が勉強したときは、B.W. カーニハン、D.M. リッチー著石田晴久訳「プログラミング言語 C　 UNIX 流プログラミング書法と作法」共立出版が唯一の本でした。この本により日本に C 言語が紹介されました。コンピュータのあるところには必ずこの本が置いてあるといわれました。ハードウエアにも興味があり、教育用の手のひらサイズのコンピュータ Raspberry PI で、linux の勉強を始めようとするなら、UNIX のコマンドがどのようにプログラミングされているか解説してあるので興味深く勉強できると思いますが、今となっては古い本で、始めてプログラミング言語を勉強しようとゆう人

には敷居が高く、歴史に興味がある人しか読む価値はありません。その後、椋田實著「はじめてのC」技術評論社が人気になりました。現在は粂井康孝著「猫でもわかるC言語プログラミング」ソフトバンクが評判が良いみたいです。「猫でもわかるC++言語プログラミング」というのもありますが、「猫でもわかるWindowsプログラミング」が昔のWindowsプログラミングを説明していて、昔、分厚い本を読んで苦労したことを思い出して現在はもう必要ないと思って敬遠していましたが、評判が良いので購入して読んでみると現在でも役に立つDLLの作り方など有用なことが簡潔に述べられています。ゲームなど時間にシビアなプログラムは昔のプログラミングが役に立つかもわかりません。ただ、この本は32bit Windowsのプログラミングの本で、時代は64ビットの時代に入っているのでこの先のことは私には分かりません。ちょこちょことWindowsのプログラミングをするだけなら別の本が良いです。兎も角、目についた本は何でもまず買ってみるものです。

私の学生時代と比べると現在は色々なことを勉強するのに本当に良い時代になりました。ただ、猛烈な勢いで世の中変わっています。若い時は勿論、何歳になっても勉強をし続けないと取り残されてしまいます。

参考文献

BJARNE STROUSTRUP : Programming Principles and Practice Using C++ Addison Wesley 2009
BJARNE STROUSTRUP : ストラウストラップのプログラミング入門
C++ によるプログラミングの原則と実践　SE 2011
Niklaus Wirth : Algorithms + Data Structures = Programs Prentice Hall 1976
林晃：C 言語逆引きハンドブック　C&R 研究所
（株）フィックスターズ：マルチコア CPU のための並列プログラミング　秀和システム
Clay Bresbears : 並行コンピューティング技法　オライリー・ジャパン　2009
昔の高等学校の数学の教科書達