C++でPENTAGOを書こう

otto4

October 6, 2017

1　序

PENTAGO とは Tomas Floden が考案した変形五目ならべで、スエーデンの Mindtwister 社が販売している Award winner in Mensa Mind Games 2006 を受賞している面白いゲームである

The game is played on a 6 × 6 board divided into four 3 × 3 sub-boards (or quadrants). Taking turns, the two players place a marble of their color (either black or white) onto an unoccupied space on the board, and then rotate one of the sub-boards by 90 degrees either clockwise or anti-clockwise. A player wins by getting five of their marbles in a vertical, horizontal or diagonal row (either before or after the sub-board rotation in their move). If all 36 spaces on the board are occupied without a row of five being formed then the game is a draw.

PENTAGO 盤は６ x ６の枡で、その四隅は 3x3 のサブボックスで回転ぢきる仕掛けになっている（自作はチョット難しい)。二人のプレーヤーは交互にプレイする。先手は黒、後手は白のマーブルを持ち、空いている枡に自分のマーブルを置き、どこか一つのサブボックスを９０°(時計または反時計まわり）回転させる。縦、横、斜めに５つマーブルが並んだら勝ちとなる、自分のプレーで相手の５連ができたら負けになり、また双方の５連が同時にできたら引き分けになる。３６枡埋まっても勝ちがない場合も引き分けになる。

ここでは碁石のことをマーブル（大理石、ピー玉）という。序盤では約 6x6x8 通りも着手かあり、その大きさでは 19 路盤の囲碁に匹敵する。

始めに、class　penta　で PENTAGO のルールの設計を説明する。（２章）次にユーザーインタフェイスを Borlandc++の　GUI で説明する。（３章）モンテカルロ法でコンピュータのプレイを書く（４章）付録でコンピュータが作製した問題集を掲げる。

2　class penta

2.1　board

class penta が局面を保持するのは黒　白　毎に 5 バイト

unsigned char A[5],B[5]

をもちいるがそのビット配置は下記のようにする

0	1	2	8	9	10
3	32	4	11	33	12
5	6	7	13	14	15
16	17	18	24	25	26
19	34	20	17	35	28
21	22	23	29	30	31

A[0] は左上隅（センターを入れない）、A[1] は右上隅、... 、A[4] はセンターの 4 ビットである。

この訳は box の回転を 1 バイトだけで済ませる配慮である。時計回りと反時計回りの 2 つのテーブルを用意しておけば play() では次のように回転している

```
  if (r<5) {
    rk=r-1;
   my.A[rk]=  rotation2[my.A[rk]];
   my.B[rk]=  rotation2[my.B[rk]];
    }
    else {
    rk=r-5;

    my.A[rk]= rotation[ my.A[rk]];
\￥    my.B[rk]= rotation[ my.B[rk]];
    }
```

2.2 勝ち型と必敗型

PENTAGO には 32 の勝ち型がある。

```
......  x.....  ......
xxxxx,  .x....  .....x
......  ..x...  .....x
......  ...x..  .....x
......  ....x.  .....x
......  ......  .....x など
```

各 winning pattern からそれにすることができる pattern を見つけることができる。 play したあとでこの pattern に一致したら負けるので必敗型と呼ぶ。必敗型は 5 バイトの pattern と 1 バイトの position 番号で表す。回転だけで 5 つ並ぶ場合に position 番号は 255 とする。

必敗型は全部で 932 個あり、これを使うと勝利の判定が早くなる。あとで述べる hash 化によって 600 ぐらいの判定にできる。

これを用いないで、すべての場合の winning pattern の判定は $6x6x8x32$ 通りになるだろう。

loosing pattern のいくつかを print したのが下記の例である。Y に置いて廻せば勝てる。

```
*  0 14 *  1 17 *  2 20 *  3 22 *  4  7 *  5 11 *  6 12 *  7 14
......  ......  ......  ......  ......  ......  ......  ......
....X.  ......  ......  ......  .X....  ...YX.  ....XY  ....X.
....Y.  ......  ......  ......  ..Y...  ......  ......  ....Y.
......  .Y....  ......  ......  .....X  ....X.  ....X.  ....X.
...XXX  .X.XXX  .XYXXX  .X.XXX  ....X.  ....X.  ....X.  ....X.
......  ......  ......  .Y....  ...X..  ....X.  ....X.  ....X.

*  8 20 *  9  2 * 10  5 * 11  7 * 12 19 * 13 18 * 14 22 * 15 28
......  ..Y...  ......  ......  ......  ......  ......  ......
......  .X....  .X....  .X....  ......  ......  ......  ......
......  ......  Y.....  ..Y...  ......  ......  ......  ......
....X.  ...X..  ...X..  ...X..  X..XXX  .XYXXX  .X....  .X....
.XY.X.  ....X.  ....X.  ....X.  Y.....  ......  .X.XX.  .X.XXY
....X.  .....X  .....X  .....X  ......  ......  .Y....  ......

* 16 27 * 17 35 * 18 34 * 19  % * 20 18 * 21 18 * 22 18 * 23 18
......  ......  ......  ......  ......  ......  ......  ......
......  ......  ......  ......  ......  ......  ......  ......
......  ......  ......  ......  ......  ......  ......  ......
.X....  .X....  .X....  .X....  .XYX..  .XY..X  XXYXX.  XXY..X
.X.YXX  .X.XYX  .Y.XXX  .X.XXX  ...X..  .....X  ......  .....X
......  ......  ......  ......  ...X..  .....X  ......  ......

* 24 18 * 25 17 * 26 17 * 27 17 * 28 20 * 29 17 * 30 17 * 31 17
......  ......  ......  ......  ......  ......  ......  ......
......  ......  ......  ......  ......  ......  ......  ......
......  ......  ......  ......  ......  ......  ......  ......
XXY...  .YXXXX  .YXX..  .YX..X  ..X...  XYXXX.  XYX..X  XYX...
...X..  ......  ...X..  .....X  ..Y...  ......  .....X  ...X..
...X..  ......  ...X..  .....X  ...XXX  ......  ......  ...X..

* 32 16 * 33 26 * 34 25 * 35 24 * 36  % * 37 29 * 38 16 * 39 31
......  ......  ......  ......  ......  ......  ......  ......
......  ......  ......  ......  ......  ......  ......  ......
......  ......  ......  ......  ......  ......  ......  ......
YXXXX.  .XXXXY  .XXXYX  .XXYXX  .XXXXX  .XXX..  YXX..X  .XX..X
......  ......  ......  ......  ......  ...X..  .....X  .....X
......  ......  ......  ......  ......  ...Y..  ......  .....Y
```

一連のテーブルは penta の最初の初期化時に setupTables() でグローバル変数として作成される。

2.3 Play と Judge

penta の主なメソッドは Play と Judge であろう。

int Play(int x,int y,int r)

は点 [x,y] にマーブルを置いて回転 r をする。x,y は　1 から 6 で　r は

r	corner	rotation
1	左上	反時計
2	右上	反時計
3	左下	反時計
4	右下	反時計
5	左上	時計
6	右上	時計
7	左下	時計
8	右下	時計

リターンコードは

rc	説明
-1	マーブルは置けない
1	黒勝ち
2	白勝ち
3	引き分け
0	正常終了

boolBack()

Play() の逆で局面を 1 手前にもどす。ユーザーインターフェースで「待った」に用いる。playout() では Back() はしようされないが、王手詰めの検索には必要。

Back() の実態は 12 バイト（ボードと step、teban）をリストアするだけでよい。囲碁の場合は局面が 361 バイトそれに島の情報も保持するには復元に play の逆手順をおこなうので厄介なものである。

boolisLoosingPattern(*intplayer*) 指定した player(1 or 2) は必敗型かどうか

を返す。

2.4 checkmate

将棋で王手の連続で勝つ手を調べる詰将棋のように王手の連続で勝つ手を探す機能が penta にある。

int GetCheckMateMove(int level)

2.5 penta のメソッド

ここで扱う penta のメソッドには次のものがある。

- penta() constructor
- penta() deconstructor
- int Play(int x,int y,int r)
  プレイする。
- bool Back()
  1 手戻す。
- int Judge1(int teban)
  teban 側の勝ちの有無
- bool Judge()
  局面は必敗型か？
- void saveBoard(unsigned char[12])
  局面を記憶する。
- void restoreBoard(unsigned char[12])
  記憶した局面に戻す。
- int getMoveList(int *zlist)
  可能な手をリストアップする。
- int getSafeMoveList(int *zlist)
  安全な手だけをリストアップする。
- int playOut()
  playout で結果を返す。

3 ユーザーインタフェイス

ツールパレットから下記のコンポーネントを Form1 に張り付ける。

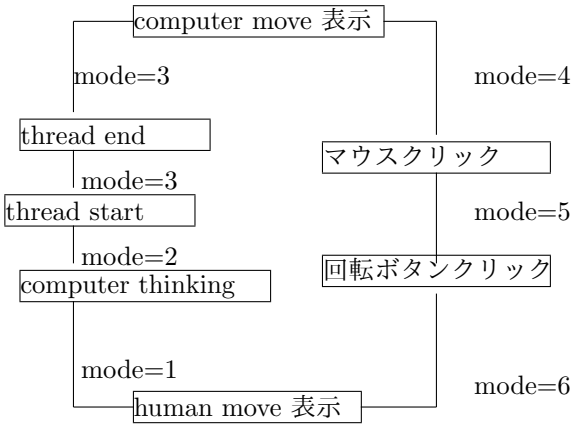
種類	パレット	個数	備考
MainMenu	standard	1	
StatusBar	Win32	1	
Timer	Syastem	1	
Button	Standard	8	
Memo	Standard	1	
OpenDialog	Dialog	1	

オブジェクトインスペクターで Form1 のイベントルーチンを 3 つ定義する。Onpaint に対応するのが paint() で、Onmousedown に対応するのが mouse() で、Ontimer で timer() が起動する。

paint() では DrawBoard() と DrawStone() を使って、現在の局面（penta *mypenta =new penta()) を描画する。最後に打ったマーブルを回転したところで△マークを付ける。

コンピュータと人間の対戦では、人はマウスクリックと回転ボタンの 2 ステップで入力を行う。

ゲームの管理はグローバル変数 mode でタイマーの監視下で切り替え動作を行う。



mode	状態
0	ゲーム開始していない
1	computerplay
2	thread 開始
3	thread 終了
4	マウス入力待ち
5	ボタン入力待ち

statuaBar は Form の最下段に作られる。statuaBar の編集をクリックして、その中に panel を 5 個追加する。その用途は:

panel	表示
0	手番
1	step 数
2	黒の残り時間
3	白の残り時間
4	playout の counter

汎用的で読者の参考になるユーザーインタフェイスとして、MainMenu から呼び出せる＜パラメータ設定＞と＜ help ＞がある。パラメータ設定はユーザー Form 型の小さな関数として TpdateParameters（）があり、help は Spawn 経由で　notepad を起動して help.txt を読ませる。

4　モンテカルロ法

モンテカルロ法の原理を簡単に説明しよう。

或る局面でのコンピュータの指し手を求める際に、すべての可能な指し手にたいして、その手をプレイしたあと、狐と狸に終局まで打たせる、これを繰り返してその指し手の勝率がもちまる。この狐と狸によるランダムな進行を playout() と呼び、この関数は勝者または引き分けを返す。

モンテカルロ法のプログラミングの要点は 2 つある。 1 つは playout の並列処理を行うためのスレッドの起動で、 2 つめは UCT 計算と呼ばれる式、playout を試みる指し手を最も効果的に選ぶ。

UCT アルゴリズムの基本式は、N 台のスロットマシンに各々n_i 回の試行がなされて、x_i の成果が得られているとき、各スロットマシンの期待値は

$$ucb = x_i + \sqrt{\frac{2 * \log n}{n_i}} \qquad n = \sum n_i$$

ここで使われている式は

$$ucb = x_i + K \times \sqrt{\frac{2 * \log n}{n_i}} \qquad n = \sum n_i$$

ここに K は　$0 < K < 1$ の定数で　実験できめる。Fukumoto 氏の Hiratuka の Igo では 0.7、　Aya は 0,5、私は 0,45 にしている。

4.1　スレッド

最近のパソコンは複数個の CPU が搭載されていて、各 CPU が 2 つのスレッドを動かせる。Intel Core i7-3930K では 12 threads が同時に動く。此の数を得るには

```
SYSTEM_INFO ggg;
GetSystemInfo(&ggg);
No_Of_CPU = ggg.dwNumberOfProcessors;
```

或るグローバル変数をスレッドでアクセス（更新）する場合には排他的制御を行う。

```
CRITICAL_SECTION montecarlo; // 変数宣言
   InitializeCriticalSection(&montecarlo);
// -------------------------
```

```
//  testcount
//---------------------------
  int testCount() {
int rc;
EnterCriticalSection(&montecarlo);
rc = remainPlayOut--;
totalcount++;
if((totalcount%100)==0) showTotalcount();
LeaveCriticalSection(&montecarlo);
return rc;
  }
```

この testcount() はスレッドがまだ playout をするかどうかお伺いをたてるときに呼び、さらに UCTtree の hash テーブルに書き込む際にも排他的制御を行う。

4.2 UCT の木構造

モンテカルロ法の実力をアップするためには、各スレッドから呼ばれる UCT は木構造にする。

UCT にレベルを設けてメモリが許す範囲（MAXLEVEL) 以内なら、playout() の替わりに UCT を呼ぶ。各 node には集計表（move,count,wincount）を持たなければならない。ここでは Hash テーブルに CHILD として組み込まれている。

```
 int UCT(penta *p,int level){
int myteban = p->teban;
int rc,i,n,move;
int k,z,zlist[300];
int brc;
float win;
p->move=0;
if(p->step>=36) return 3;
EnterCriticalSection(&montecarlo);
// ----------------critical-start----------------------
NODE *node = myhash->HashSearch(p->my.A);
if (node->child_num == 0) {
CHILD*pc;
n = p->getSafeMoveList(zlist);
for (i = 0; i < n;i++) {

pc = &node->child[node->child_num++];
pc->move = zlist[i];
pc->games = 0;
```

```
pc->win = 0;
}
}

// DoPrintf("cn=%d", node->child_num);
LeaveCriticalSection(&montecarlo);
  // mtx->ReleaseMutex();
if(node->child_num ==0){
return 3-p->teban;
}
// ----------------critical-end----------------------
int loop;
int create_new_node_limit = 60;
n = node->child_num;
int upper_num = n;
int select = -1;
const int DEPTHMAX = 8;
const flost factor=0,44;
double max_value = -10000.;
double uct_value;
for (loop = 0; loop < n; loop++) {
CHILD *pc = &node->child[loop];
if (pc->games) {
double rate = 0;
if (pc->games)rate = pc->win / ((double)pc->games);
double utc = factor * sqrt
(log((double)node->games_sum) / pc->games);
uct_value = rate + utc;
}
else {
uct_value = 10000 + rand() % 100;
}

if (uct_value > max_value)
max_value = uct_value, select = loop;
}
CHILD *pc = &node->child[select];
move= pc->move;
rc = p->Play(move/100,(move/10)%10,move%10);
if (rc)return rc;

if (pc->games < create_new_node_limit ||level>= DEPTHMAX ||
p->step>30 ) {  rc = p->playOut();}
else { rc = UCT(p, level + 1); }
// update node--------------------
p->move = move;
```

```
//  --critical-start--
EnterCriticalSection(&montecarlo);
win = 0;
if(rc == myteban) win=1;
else if(rc==3) win=0.5;
pc->games++;
pc->win+=win;
node->games_sum++;
LeaveCriticalSection(&montecarlo);
// ---critical-end-;
return rc;

}
```

penta のメトッドとして書かれている playout() は

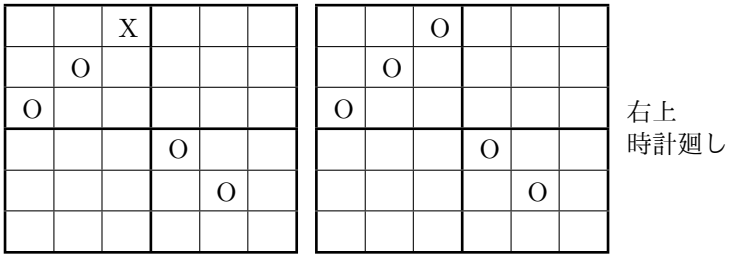
```
int penta::playOut(){
int   zw[300];
int wm,z, x, y, r, n, rc, k,t;

if (step == 36) {
maxlevel = step; return 3;
}
n = getMoveList(zw);
t = teban;
while (n > 0){
k = rand() % n;
z = zw[k];
rc = Play(z / 100, (z / 10) % 10, z %10);
if (rc==t|| rc==3){
maxlevel = step;
return rc;
}
else if(rc==0 &&Judge()) rc=-1;
if (rc == 0) break;
zw[k] = zw[n - 1];
n--;
Back();
}

if (n == 0)  {
maxlevel = step;
return 3 - teban;
}

 //統計的に playiut の深さ調べるため
```

```
if (rc > 0) { maxlevel = step; return rc;  }
return   playOut();
}
```

5 序盤戦術

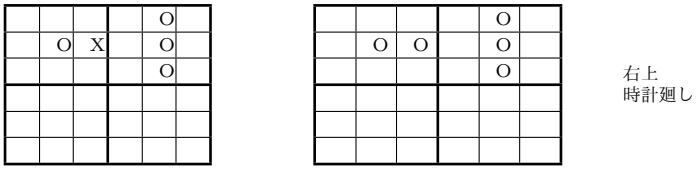
5.1 Monica's five

最初の 2 手は対角の BOX のセンターに打つ。



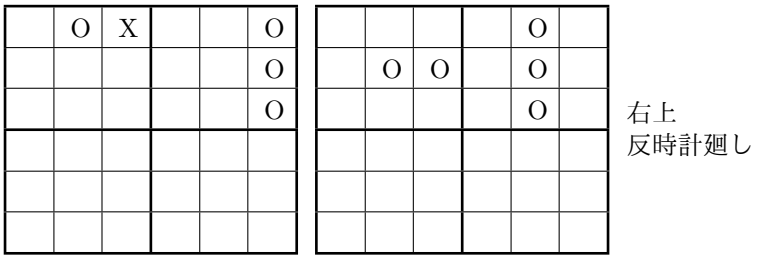
5.2 Middle five

最初の 2 手は隣り合う BOX のセンターに打つ。



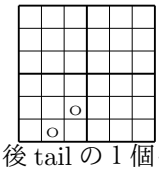
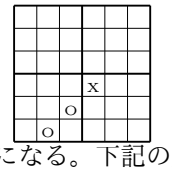
5.3 Straight five

最初の 3 手は辺の三連に置く。

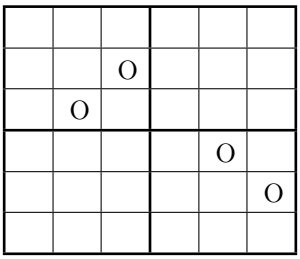
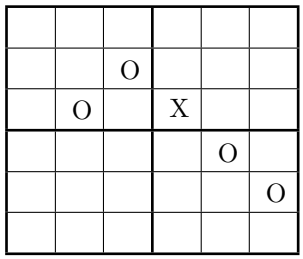


5.4 Triple power

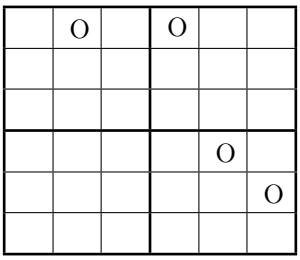
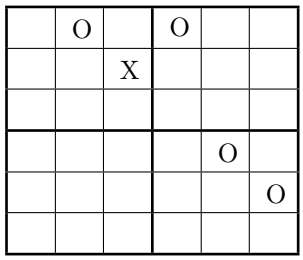
こらは最も強力な作戦で、騙されやすく、変化も多い。はじめに、　top　と呼ぶ 2 個を置く。次に pivot と呼ぶ X を置く。

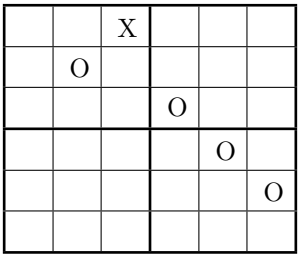
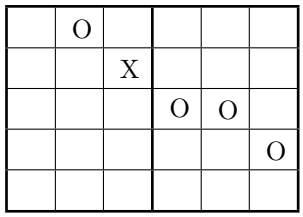
その後 tail の 1 個を置けば王手になる。下記の勝ちパターンがある。

左上
反時計廻し

右上
反時計廻し

左上
時計廻し

6 pthread で genmove

BorlandC++には　Boost を install しないと pthread は使えない。PENTAGO のコンソールアプリケーションとして VC13 でかいた PENTAGO は pthread を用いている。pthread は thread にパラメータを容易に渡せて、thread の終了を Join() で待てるのでプログラムがすっきりかける。

```
//thread function
static void threadFunc(Object^ state)
{
```

```
State^ s = dynamic_cast<State^>(state); //キャストが必要
penta *p = new penta;
int rc0, rc, move, teban;
float win;
p->restoreBoard(s->gg);
teban = p->teban;

//DoPrintf("threadfunc n=%d step=%d", n,p->step);
while (getnext(s->mtx)){
p->restoreBoard(s->gg);
rc = UCT(p, s->mtx, 0);
}
}

int genmove(unsigned char board[12]){
SYSTEM_INFO ggg;
int i, ii;
GetSystemInfo(&ggg);
totalplayoutcount=0;
totalLengthCount=0;
    myhash = new HASH(0x4000);
//List<int>^ primes; //結果を格納するリストへのハンドル
int No_Of_CPU = ggg.dwNumberOfProcessors;
DateTime^ start = DateTime::Now; //開始時間
Mutex^ mtx = gcnew Mutex; //ロック用のオブジェクト
List<Thread^>^  threadlist = gcnew List<Thread^>;
int NCPU = No_Of_CPU;
totalLengthCount = 0;
playoutcount = 0;
nmove = n;
DoPrintf("modws passed to genmove moves= %d", n);
//thread 開始
for (ii = 0; ii < NCPU; ii++){
threadlist->Add(gcnew Thread(gcnew ParameterizedThreadStart(threadFunc)));
threadlist[ii]->Start(gcnew State(board, mtx));
}
//thread の終了を待つ
for (ii = 0; ii < NCPU; ii++){ threadlist[ii]->Join(); }
DateTime^ finish = DateTime::Now; //終了時間
TimeSpan duration = finish->Subtract(*start); //経過時間の計算
int t1 = duration.TotalSeconds;
DoPrintf("time=%d sec", t1);
int move = getBestMove(board,myhash);
if (totalplayoutcount)
DoPrintf("mean depth=%d total=%d", totalLengthCount / totalplayoutcount,        totalpla
delete myhash;
```

```
return move;
}
int getBestMove(unsigned char *board){
NODE *node = myhash->HashSearch(board);
no_of_moves=node->child_num ;
int n=  no_of_moves;
CHILD*pc=node->child;
qsort(pc, n, sizeof(CHILD), comp8);
int k;
k = n;
if (n > 5) k = 5;
for (int i = 0; i < k; i++){
float winrate=0;
if( pc[i].games>0) winrate= pc[i].win/ pc[i].games;
DoPrintf("i=%d move=%3d rate=%5.3f games=%4d win=%4.1f",
i + 1, pc[i].move, winrate, pc[i].games, pc[i].win);

}
 return pc[0].move;
}
```

7 Borlandc++のビルド

PENTAGO2 フォルダーにはつぎのファイルがある.

No	filename	extention	note
1	pentagomain	cpp,h,dfm	FORM
2	penta	cpp,h	
3	hash	cpp,h	
4	playoutthread3	cpp,h	
5	computerplay	cpp	
6	optionsetup	cpp,h,dfm	
7	pentago2	cpp,res,cbproj	project

8 VC13 のビルド

VC13 で CLR のコンソールアプリケーションを作製でる。
PENTAGO1 フォルダーにはつぎのファイルがある。

No	filename	extention	note
1	pentago1main	cpp	FORM
2	penta	cpp,h	
3	hash	cpp,h	
4	computerplay	cpp	
5	pentago1	ln	project