```
#include <cmath>
#include <ctime>
#include <queue>
#include <string>
#include <cstring>
#include <cstdlib>
#include <iostream>
#include <algorithm>
#define inf 100000
using namespace std;
#define W 10
#define H 20
#define N 11
double R;
int dfsW;
int K;
int PH;
int nextTypeForColor[2];
int CURH[2];
int currBotColor;
int enemyColor;
int RECID;
int REC[105][H + 2][W + 2], REC2[105][H + 2][W + 2];
int gridInfo[2][H + 2][W + 2];
int trans[2][4][W + 2];
int transCount[2];
int maxHeight[2];
int elimTotal[2];
int elimCombo[2];
int turnID, BType;
int BForEnemy, finalX, finalY, finalO;
double VAL[W + 2] = {0, 0.6, 0.4, 0.2, 0.1, 0, 0, 0.1, 0.2,
0.4, 0.6, 0};
const int elimBonus[] = {0, 1, 3, 5, 7 };
double C[8][N] = {
     {10,-0.66,-1.2,-1.5,-4.3,-5.5,-5.3,-11,0.95,1.3,0.67},
{10,-0.82,-2.5,-1.4,-1.2,-2.5,-3.8,-12,1,2.3,1.2},
{10,-1.1,-2.5,-2.3,-1.1,-2.2,-2.7,-7.4,1,2.8,0.39},
{10,-1.6,-3.6,-2.1,-1.2,-2,-2.8,-6.4,1.1,4.6,0.6},
{10,-0.7,-1.4,-1.6,-4.5,-6,-5.3,-13,1.1,1.4,0.72},
{10,-0.88,-2.8,-1.4,-1.2,-2.4,-5,-12,1,2.4,1.3},
{10,-0.96,-2.1,-2.4,-1.2,-2.1,-2.2,-6.5,0.95,2.9,0.38},
{10,-1.4,-3.4,-1.8,-1.1,-1.9,-2.6,-5.5,1,4,0.54},
};
//double A[] = {VC, RT, MH, WS, NH[0], NH[1], NH[2],
(double)max(C4 - 1, 0)};
int CCnt[2][7];
int MN[2], MX[2];

const int BShape[7][4][8] = {
     { { 0,0,1,0,-1,0,-1,-1 },{ 0,0,0,1,0,-1,1,-1 },
{ 0,0,-1,0,1,0,1,1 },{ 0,0,0,-1,0,1,-1,1 } },
     { { 0,0,-1,0,1,0,1,-1 },{ 0,0,0,-1,0,1,1,1 },
{ 0,0,1,0,-1,0,-1,1 },{ 0,0,0,1,0,-1,-1,-1 } },
     { { 0,0,1,0,0,-1,-1,-1 },{ 0,0,0,1,1,0,1,-1 },
{ 0,0,-1,0,0,1,1,1 },{ 0,0,0,-1,-1,0,-1,1 } },
     { { 0,0,-1,0,0,-1,1,-1 },{ 0,0,0,-1,1,0,1,1 },
{ 0,0,1,0,0,1,-1,1 },{ 0,0,0,1,-1,0,-1,-1 } },
     { { 0,0,-1,0,0,1,1,0 },{ 0,0,0,-1,-1,0,0,1 },
{ 0,0,1,0,0,-1,-1,0 },{ 0,0,0,1,1,0,0,-1 } },
     { { 0,0,0,-1,0,1,0,2 },{ 0,0,1,0,-1,0,-2,0 },
{ 0,0,0,1,0,-1,0,-2 },{ 0,0,-1,0,1,0,2,0 } },
     { { 0,0,0,1,-1,0,-1,1 },{ 0,0,-1,0,0,-1,-1,-1 },
{ 0,0,0,-1,1,-0,1,-1 },{ 0,0,1,0,0,1,1,1 } }
};

const int rotateBlank[7][4][10] = {
     { { 1,1,0,0 },{ -1,1,0,0 },{ -1,-1,0,0 },{ 1,-1,0,0 } },
     { { -1,-1,0,0 },{ 1,-1,0,0 },{ 1,1,0,0 },{ -1,1,0,0 } },
     { { 1,1,0,0 },{ -1,1,0,0 },{ -1,-1,0,0 },{ 1,-1,0,0 } },
     { { -1,-1,0,0 },{ 1,-1,0,0 },{ 1,1,0,0 },{ -1,1,0,0 } },
     { { -1,-1,-1,1,1,1,0,0 },{ -1,-1,-1,1,1,-1,0,0 },
{ -1,-1,1,1,1,-1,0,0 },{ -1,1,1,1,1,-1,0,0 } },
     { { 1,-1,-1,1,-2,1,-1,2,-2,2 } ,
{ 1,1,-1,-1,-2,-1,-1,-2,-2,-2 } ,{ -1,1,1,-1,2,-1,1,-2,2,-2 }
,{ -1,-1,1,1,2,1,1,2,2,2 } },
     { { 0,0 },{ 0,0 } ,{ 0,0 } ,{ 0,0 } }
};

int G[H + 2][W + 2];
int cnt[W + 2], dep[W + 2], well[W + 2];
int val[H + 2][W + 2];
const int xx[] = {0, 0, 1, -1};
const int yy[] = {1, -1, 0, 0};

void retreated(int color){
    if(!RECID)return;
    for(int y = H; y >= 1; y--)
        for(int x = 0; x <= W + 1; x++)
        {
            gridInfo[color][y][x] = REC[RECID][y][x];
            val[y][x] = REC2[RECID][y][x];
        }
    RECID--;
}

class Tetris
{
public:
    const int BType;   // 标记方块类型的序号 0~6
    int BX;            // 旋转中心的x轴坐标
    int BY;            // 旋转中心的y轴坐标
    int BO;        // 标记方块的朝向 0~3
    const int(*shape)[8]; // 当前类型方块的形状定义
    int color;
    Tetris(int t, int color) : BType(t), shape(BShape[t]),
color(color){ }
    Tetris &set(int x, int y, int o){
        BX = x, BY = y, BO = o;
        return *this;
    }
    // 判断当前位置是否合法
    bool isValid(int x = -1, int y = -1, int o = -1){
        x = x == -1 ? BX : x;
        y = y == -1 ? BY : y;
        o = o == -1 ? BO : o;
        if(o < 0 || o > 3)
            return false;

        int i, tmpX, tmpY;
        for(i = 0; i < 4; i++){
            tmpX = x + shape[o][2 * i];
            tmpY = y + shape[o][2 * i + 1];
            if(tmpX < 1 || tmpX > W ||
               tmpY < 1 || tmpY > H ||
               gridInfo[color][tmpY][tmpX] != 0)
                return false;
        }
        return true;
    }
    // 判断是否落地
    bool onGround(){
        if(isValid() && !isValid(-1, BY - 1))
            return true;
        return false;
    }
    // 将方块放置在场地上
    void place(bool rec){
        PH = 100;
        if(rec)
        {
            ++RECID;
            for(int y = H; y >= 1; y--)
                for(int x = 0; x <= W + 1; x++)
                {
                    REC[RECID][y][x] = gridInfo[color][y]
[x];
                    REC2[RECID][y][x] = val[y][x];
                }
        }
        if(!onGround())return;
        int i, tmpX, tmpY;
        for(i = 0; i < 4; i++)
        {
            tmpX = BX + shape[BO][2 * i];
            tmpY = BY + shape[BO][2 * i + 1];
            PH = min(PH, tmpY);
            gridInfo[color][tmpY][tmpX] = 2;
        }
    }
    // 检查能否逆时针旋转自己到o
    bool rotation(int o){
        if(o < 0 || o > 3)
            return false;
        if(BO == o)
            return true;
        int fromO = BO;
        while(true)
        {
            if(!isValid(-1, -1, fromO))
                return false;
            if(fromO == o)
                break;
          for (int i = 0; i < 5; i++) {
              int blankX = BX + rotateBlank[BType][fromO]
[2 * i];
              int blankY = BY + rotateBlank[BType][fromO]
[2 * i + 1];
              if (blankX == BX && blankY == BY)
                  break;
              if (gridInfo[color][blankY][blankX] != 0)
                  return false;
           }
            fromO =(fromO + 1) % 4;
        }
        return true;
    }
};

// 围一圈护城河
void init()
{
    int i;
    for(i = 0; i < H + 2; i++)
    {
        gridInfo[1][i][0] = gridInfo[1][i][W + 1] = -2;
        gridInfo[0][i][0] = gridInfo[0][i][W + 1] = -2;
    }
    for(i = 0; i < W + 2; i++)
    {
        gridInfo[1][0][i] = gridInfo[1][H + 1][i] = -2;
        gridInfo[0][0][i] = gridInfo[0][H + 1][i] = -2;
    }
}

namespace Util
{
    // 检查能否从场地顶端直接落到当前位置
    bool checkDirectDropTo(int color, int BType, int x, int
y, int o)
    {
        auto &def = BShape[BType][o];
        for (; y <= H; y++)
            for (int i = 0; i < 4; i++)
            {
                int _x = def[i * 2] + x, _y = def[i * 2 +
1] + y;
                if (_y > H)
                    continue;
                if (_y < 1 || _x < 1 || _x > W ||
gridInfo[color][_y][_x])
                    return false;
            }
        return true;
    }
    // 消去行
    void eliminate(int color)
    {
        int &count = transCount[color] = 0;
        int i, j, emptyFlag, fullFlag, firstFull = 1,
hasBonus = 0;
        maxHeight[color] = H;
        for(i = 1; i <= H; i++)
        {
            emptyFlag = 1;
            fullFlag = 1;
            for(j = 1; j <= W; j++)
            {
                if(gridInfo[color][i][j] == 0)
                    fullFlag = 0;
```

```
                  else
                      emptyFlag = 0;
              }
              if(fullFlag)
              {
                if (firstFull && ++elimCombo[color] >= 3)
                  {
                      // 奖励行
                      for (j = 1; j <= W; j++)
                          trans[color][count][j] =
gridInfo[color][i][j] == 1 ? 1 : 0;
                      count++;
                      hasBonus = 1;
                  }
                firstFull = 0;
                  for(j = 1; j <= W; j++)
                  {
                      // 注意这里只转移以前的块，不包括最后一次落下
的块（“撤销最后一步”）
                   trans[color][count][j] = gridInfo[color]
[i][j] == 1 ? 1 : 0;
                      gridInfo[color][i][j] = 0;
                  }
                  count++;
              }
              else if(emptyFlag)
              {
                  maxHeight[color] = i - 1;
                  break;
              }
              else
                for (j = 1; j <= W; j++)
                  {
                      gridInfo[color][i - count + hasBonus]
[j] =
                          gridInfo[color][i][j] > 0 ? 1 :
gridInfo[color][i][j];
                      if (count)
                          gridInfo[color][i][j] = 0;
                  }
```

```
        }
       if (count == 0)
            elimCombo[color] = 0;
        maxHeight[color] -= count - hasBonus;
       elimTotal[color] += elimBonus[count];
    }

    // 转移双方消去的行，返回-1表示继续，否则返回输者
    int transfer()
    {
        int color1 = 0, color2 = 1;
        if(transCount[color1] == 0 && transCount[color2] ==
0)
            return -1;
        if(transCount[color1] == 0 || transCount[color2] ==
0)
        {
            if(transCount[color1] == 0 &&
transCount[color2] > 0)
                swap(color1, color2);
            int h2;
            maxHeight[color2] = h2 = maxHeight[color2] +
transCount[color1];
            if(h2 > H)
                return color2;
            int i, j;

            for(i = h2; i > transCount[color1]; i--)
                for(j = 1; j <= W; j++)
                    gridInfo[color2][i][j] =
gridInfo[color2][i - transCount[color1]][j];

            for(i = transCount[color1]; i > 0; i--)
                for(j = 1; j <= W; j++)
                    gridInfo[color2][i][j] = trans[color1]
[i - 1][j];
            return -1;
        }
        else
        {
            int h1, h2;
            maxHeight[color1] = h1 = maxHeight[color1] +
transCount[color2];//从color1处移动count1去color2
            maxHeight[color2] = h2 = maxHeight[color2] +
transCount[color1];

            if(h1 > H) return color1;
            if(h2 > H) return color2;

            int i, j;
            for(i = h2; i > transCount[color1]; i--)
                for(j = 1; j <= W; j++)
                    gridInfo[color2][i][j] =
gridInfo[color2][i - transCount[color1]][j];

            for(i = transCount[color1]; i > 0; i--)
                for(j = 1; j <= W; j++)
                    gridInfo[color2][i][j] = trans[color1]
[i - 1][j];

            for(i = h1; i > transCount[color2]; i--)
                for(j = 1; j <= W; j++)
                    gridInfo[color1][i][j] =
gridInfo[color1][i - transCount[color2]][j];

            for(i = transCount[color2]; i > 0; i--)
                for(j = 1; j <= W; j++)
                    gridInfo[color1][i][j] = trans[color2]
[i - 1][j];

            return -1;
        }
    }

    // 打印场地用于调试
    void printField()
    {
#ifndef _BOTZONE_ONLINE
        static const char *i2s[] = {
            "~~",
            "~~",
            "  ",
            "[]",
            "##"
        };
        cout << "~~: 墙, []: 块, ##: 新块" << endl;
        for(int y = H + 1; y >= 0; y--)
        {
            for(int x = 0; x <= W + 1; x++)
                cout << i2s[gridInfo[0][y][x] + 2];
            for(int x = 0; x <= W + 1; x++)
                cout << i2s[gridInfo[1][y][x] + 2];
            cout << endl;
        }
#endif
    }
}

void bfs(int BType, int color){
    Tetris B(BType, color);
    memset(val, 0, sizeof(val));
    queue<Tetris> q;
    for(int y = 1; y <= H; y++)
        for(int x = 1; x <= W; x++)
            for(int o = 0; o < 4; o++)
            {
                if(B.set(x, y, o).isValid() &&
                   Util::checkDirectDropTo(color, BType, x,
y, o))
                {
                    val[y][x] |= (1 << o);
                    q.push(Tetris(BType, color).set(x, y,
o));
                }
            }
    while(!q.empty()){
        Tetris u = q.front(); q.pop();
        if(u.rotation((u.BO + 1) % 4))
            if(!(val[u.BY][u.BX] & (1 << ((u.BO + 1) %
4))))
```

```
            {
                Tetris v = Tetris(BType, color).set(u.BX,
u.BY, (u.BO + 1) % 4);
                if(!v.isValid())continue;
                q.push(v);
                val[v.BY][v.BX] |= 1 << v.BO;
            }
        for(int i = 0; i < 4; i++)
        {
            int x = u.BX + xx[i], y = u.BY + yy[i];
            Tetris v = Tetris(BType, color).set(x, y,
u.BO);
            if(v.isValid())
                if(!(val[y][x] & (1 << v.BO)))
                {
                    q.push(v);
                    val[y][x] |= 1 << v.BO;
                }
        }
    }
    for(int y = 1; y <= H; y++)
        for(int x = 1; x <= W; x++)
            for(int o = 0; o < 4; o++)
                if(val[y][x] & (1 << o))
                {
                    if(BType == 6 && o)val[y][x] -= (1 <<
o);
                    if((BType == 2 || BType == 3 || BType
== 5) && o <= 1)
                        val[y][x] -= (1 << o);
                }
}
void add(int c, int x, int val)
{
    CCnt[c][x] += val;
    MX[c] = *max_element(CCnt[c], CCnt[c] + 7);
    MN[c] = *min_element(CCnt[c], CCnt[c] + 7);
}
bool check(int c, int x)
{
    return (MX[c] - MN[c] < 2 || CCnt[c][x] != MX[c]);
}
int WAIT(int color, int p)
{
    int ans = 0;
    for(int i = 0; i < 7; i++)
        if(i != p)
            ans += CCnt[color][p] + 2 - CCnt[color][i];
    return ans;
}
double f(int color, double *p)
{
    double ans = 0;
    for(int i = 0; i < N - 2; i++)
        ans += C[K][i] * p[i];
    return ans;
}
double Evaluate(int p, bool debug = 0)
{
    memset(cnt, 0, sizeof(cnt));
    memset(well, -1, sizeof(well));
    memset(dep, 0, sizeof(dep));
    int W0 = WAIT(p, 0), W1 = WAIT(p, 1), W5 = WAIT(p, 5);
    if(debug)cout << W5 << endl;
    double RT = 0, NH[3] = {0}, WS = 0, VC = 0, AV = 0, MH =
0;
    for(int y = H; y >= 0; y--)
        for(int x = 0; x <= W + 1; x++)
            if(gridInfo[p][y][x] == 0)
                G[y][x] = 0;
             else
             {
                 if(!MH && x != 0 && x != W + 1)MH = y;
                 G[y][x] = 1;
             }
    for(int y = H; y >= 0; y--)
    {
        bool havH = 0;
        for(int x = 0; x <= W; x++)
            if(G[y][x] != G[y][x + 1])RT++;
        for(int x = 1; x <= W; x++)
        {
            if(G[y][x] == 1)
                cnt[x]++;
            else if(cnt[x])
            {
                havH = 1;
                G[y][x] = 2;
                if(cnt[x] <= 3)NH[cnt[x] - 1]++;
                else NH[2]++;
            }
        }
        well[1] = 1; well[W] = 0;
        for(int x = 1; x <= W; x++)
            if(G[y][x] == 0 && !cnt[x])
            {
                if(well[x] == -1)
                {
                    if(!G[y][x - 1] && G[y][x + 1] && x !=
W)well[x] = 0;
                    if(!G[y][x + 1] && G[y][x - 1] && x !=
1)well[x] = 1;
                }
                if(G[y][x - 1] && G[y][x + 1])dep[x]++;
            }
        if(!havH && y)
        {
            VC++;
            for(int x = 1; x <= W; x++)
                if(G[y][x] == 1)cnt[x]++;
        }
    }
    for(int x = 1; x <= W; x++)
        if(dep[x])
        {
            double tmp = dep[x] * (dep[x] + 1) / 2;
            if(turnID <= 35)
            {
                if(dep[x] == 2)
                {
                    if(well[x] == 0 && W0 > 10)WS += tmp /
2;
                    if(well[x] == 1 && W1 > 10)WS += tmp /
2;
                    if(well[x] == 0 && W0 <= 4)dep[x] = 0,
W0 += 6;
                    if(well[x] == 1 && W1 <= 4)dep[x] = 0,
W1 += 6;
                    if(well[x] == -1 && min(W0, W1) <=
6)dep[x] = 0;
                }
                if(dep[x] > 2)
                {
                    if(W5 > 10)WS += tmp;
                    if(W5 <= 4)W5 += 6, dep[x] = 0;
                }
            }
            WS += dep[x] * (dep[x] + 1) / 2;
        }
    int C4 = 0;
    for(int x = 1; x <= W; x++)
        if(dep[x] >= 3)C4++;
    if(MH == 20)MH += 5;
    if(MH == 19)MH += 3;
    if(MH == 18)MH += 1;
    MH += 0.8 * PH;
    double tot = 0;
    for(int x = 1; x <= W; x++)
        tot += cnt[x] * VAL[x] / 2;
    double A[] = {VC, RT, MH, WS, NH[0], NH[1], NH[2],
(double)max(C4 - 1, 0), tot};
    return f(p, A);
}
int top, st[505][10], tmp[10];
struct data{
    int y, x, o;
    double tmp;
};
bool operator<(data a, data b){
    return a.tmp > b.tmp;
```

```
}
double solve(int Btype, int color, int dep, int D, int T,
double mn)
{
    Tetris B(Btype, color);
    bfs(Btype, color);
    double mx = -2 * inf;
    int tx, ty, to;
    int p = 0;
    data ve[60];
    int vtop = 0;
    for(int y = 1; y <= H; y++)
        for(int x = 1; x <= W; x++)
            for(int o = 0; o < 4; o++)
            {
                if(B.set(x, y, o).onGround() && (val[y][x]
& 1 << o))
                {
                    add(color, Btype, 1);
                    B.set(x, y, o).place(1);

                    Util::eliminate(color);
                    double score = C[K][N - 2] *
elimCombo[color];
                    if(CURH[color] >= 18)score *= 5;
                    else if(CURH[color ^ 1] >= 18)
                        score *= 5;
                    double tmp = score;
                    if(dep < D)tmp += Evaluate(color) /
C[K][N - 1] * 2;
                    else tmp += Evaluate(color);
                    ve[vtop++] = ((data){y, x, o, tmp});
                    if(dep == D && tmp > mx)
                    {
                        mx = tmp;
                        tx = x, ty = y, to = o;
                    }
                    retreated(color);
                    add(color, Btype, -1);
                    if(mx > mn)return mx;/* 剪枝 */
                    }
               }
     if(!vtop)
          return -inf + dep * 1000;
     if(dep < D)
     {
          sort(ve, ve + vtop);
          for(int i = 0, j = 0; i < vtop; i++)
          {
               add(color, Btype, 1);
               B.set(ve[i].x, ve[i].y, ve[i].o).place(1);
               Util::eliminate(color);
               double tmp = ve[i].tmp + solve(st[T][dep],
color, dep + 1, D, T, mn - ve[i].tmp);
               if(tmp > mx)
               {
                    mx = tmp;
                    tx = ve[i].x, ty = ve[i].y, to = ve[i].o;
               }
               retreated(color);
               add(color, Btype, -1);
               if(mx > mn)return mx;/* 剪枝 */
               j++;
               if(j == dfsW)break;
          }
     }
     return mx;
}
int DFN[] = {6, 5, 3, 2, 4, 1, 0};
void dfs(int color, int dep, int D)
{
     if(dep == D)
     {
          top++;
          for(int i = 0; i < D; i++)
               st[top][i] = tmp[i];
          return;
     }
     for(int i = 0; i < 7; i++)
          if(check(color, DFN[i]))
```

```
        {
            add(color, DFN[i], 1);
           tmp[dep] = DFN[i];
            dfs(color, dep + 1, D);
            add(color, DFN[i], -1);
        }
}
int sel(int color)
{
    dfsW = 4;
   if(turnID <= 15)K = 4;
    else if(turnID <= 25)K = 5;
    else if(turnID <= 38)K = 6;
    else K = 7;
    top = 0; int D = 2;
    dfs(color, 0, D);
    double mn = inf; int ans = 0;
    for(int i = top; i; i--)
    {
        add(color, nextTypeForColor[color], -1);
        double tmp = solve(nextTypeForColor[color], color,
0, D + 1, i, mn);
        add(color, nextTypeForColor[color], 1);
        if(tmp <= mn)
        {
            mn = tmp;
            ans = st[i][0];
        }
    }
    return ans;
}
void run(int Btype, int color)
{
   if(turnID <= 15)K = 0;
    else if(turnID <= 25)K = 1;
    else if(turnID <= 38)K = 2;
    else K = 3;
    double mx = -2147483647;
    top = 0; int D = 2;
    dfs(color, 0, D);
    dfsW = 4;
    Tetris B(Btype, color);
    bfs(Btype, color);
    data ve[60];
    int vtop = 0;
    for(int y = 1; y <= H; y++)
        for(int x = 1; x <= W; x++)
            for(int o = 0; o < 4; o++)
            {
                if(B.set(x, y, o).onGround() && (val[y][x]
& (1 << o)))
                {
                    B.set(x, y, o).place(1);
                    Util::eliminate(color);
                    double score = C[K][N - 2] *
elimCombo[color];
                    if(CURH[color] >= 18)score *= 5;
                    else if(CURH[color ^ 1] >= 18)
                        score *= 5;
                    score += Evaluate(color) / C[K][N - 1]
* 2; // 与贪心取平均
                    ve[vtop++] = ((data){y, x, o, score});
                    retreated(color);
                }
            }
    sort(ve, ve + vtop);
    for(int i = 0, j = 0; i < vtop; i++)
    {
        double mn = inf;
        B.set(ve[i].x, ve[i].y, ve[i].o).place(1);
        Util::eliminate(color);
        for(int t = 1; t <= top; t++)
        {
            double tmp = ve[i].tmp + solve(st[t][0], color,
0, D, t, mn - ve[i].tmp);
            mn = min(mn, tmp);
            if(mn <= mx)break;
        }
        retreated(color);
        if(mn > mx)
```

```
          {
               mx = mn;
               finalX = ve[i].x, finalY = ve[i].y, finalO =
ve[i].o;
          }
          j++;
          if(j == 6)break;
     }
}
void run2(int Btype, int color)
{
    K = 0;
     double mx = -2147483647;
     top = 0; int D = 2;
     dfs(color, 0, D);
    if(top >= 10)
     {
          top = 0, D = 1;
          dfs(color, 0, D);
     }
     dfsW = 4;
     Tetris B(Btype, color);
     bfs(Btype, color);
     data ve[60];
     int vtop = 0;
     for(int y = 1; y <= H; y++)
          for(int x = 1; x <= W; x++)
               for(int o = 0; o < 4; o++)
               {
                    if(B.set(x, y, o).onGround() && (val[y][x]
& (1 << o)))
                    {
                         B.set(x, y, o).place(1);
                         Util::eliminate(color);
                         double score = C[K][N - 2] *
elimCombo[color];
                         if(CURH[color] >= 18)score *= 5;
                         else if(CURH[color ^ 1] >= 18)
                              score *= 5;
```

```
                        score += Evaluate(color) / C[K][N -
1]; // 与贪心取平均
                        ve[vtop++] = ((data){y, x, o, score});
                        retreated(color);
                    }
                }
    sort(ve, ve + vtop);
    for(int i = 0, j = 0; i < vtop; i++)
    {
        double mn = inf , tot = 0;
        B.set(ve[i].x, ve[i].y, ve[i].o).place(1);
        Util::eliminate(color);
        for(int t = 1; t <= top; t++)
        {
            double tmp = ve[i].tmp + solve(st[t][0], color,
0, D, t, inf);
            tot += tmp;
            mn = min(mn, tmp);
        }
        mn = mn + tot / top * R;
        retreated(color);
        if(mn > mx)
        {
            mx = mn;
            finalX = ve[i].x, finalY = ve[i].y, finalO =
ve[i].o;
        }
        j++;
        if(j == 8)break;
    }
}
int main()
{
    srand(time(0));
    R = (double)rand() / RAND_MAX + 0.5;
    istream::sync_with_stdio(false);
    init();
    cin >> turnID;
    cin >> BType >> currBotColor;
    enemyColor = 1 - currBotColor;
```

```
	nextTypeForColor[0] = BType;
	nextTypeForColor[1] = BType;
	add(0, BType, 1);
	add(1, BType, 1);
	for(int i = 1; i < turnID; i++)
	{
		int currTypeForColor[2] = { nextTypeForColor[0],
nextTypeForColor[1] };
		int x, y, o;
		cin >> BType >> x >> y >> o;
		Tetris myB(currTypeForColor[currBotColor],
currBotColor);
		myB.set(x, y, o).place(0);
		add(enemyColor, BType, 1);
		nextTypeForColor[enemyColor] = BType;
		cin >> BType >> x >> y >> o;
		Tetris enemyB(currTypeForColor[enemyColor],
enemyColor);
		enemyB.set(x, y, o).place(0);
		add(currBotColor, BType, 1);
		nextTypeForColor[currBotColor] = BType;
		Util::eliminate(0);
		Util::eliminate(1);
		Util::transfer();
	}
//	cout << Evaluate(currBotColor, 1);
	for(int i = 0; i < 2; i++)
		for(int y = H; y >= 1 && !CURH[i]; y--)
			for(int x = 1; x <= W; x++)
				if(gridInfo[i][y][x])
					CURH[i] = y;
	//决策
	if(turnID > 15)
		run(nextTypeForColor[currBotColor], currBotColor);
	else run2(nextTypeForColor[currBotColor], currBotColor);
	Tetris B(nextTypeForColor[currBotColor], currBotColor);
	B.set(finalX, finalY, finalO).place(1);
	Util::printField();
	Util::eliminate(currBotColor);
	Util::transfer();
```

```
    if(turnID == 1)BForEnemy = rand() % 7;
     else BForEnemy = sel(enemyColor);
     cout << BForEnemy << " " << finalX << " " << finalY << "
" << finalO << endl;
     return 0;
}
```