附录：

在此附上本机版本的源代码。

1.头文件 Amazon.h

```
#ifndef AMAZON_H
#define AMAZON_H

#include <queue>
#include <cstdio>

typedef std :: pair <int, int> node;

class amazon{
public:
    const static int N = 8;
    const static int inf = 1e9;
    const static int TOTAL = 5e4 + 5;
    constexpr static double Ec = 0.08; //设置先手优势系数

    int Num[6], TurnCount, curBotColor, map[N][N], level;

    amazon();

    //博弈部分
    void Initialize ();
    bool Judge(); //判断执子
    void DecisionMaking();

    //存档记录部分
    int ReCordm, ReCordlen, ReCordlevel, ReCorda[100][6], ReCordflag;
    void ReCordInitialize ();
    void ReCordRecord();
    void ReCordWithDraw();
    void ReCordLoad();

    //输入部分
    int expected[N][N];
    void expect(const int &x, const int &y, int &stepCount);

private:
    std :: queue <node> que;
    FILE* fp;
```

```
    int gox[N] = {1, -1, -1, 0, 1, -1, 0, 1};
    int goy[N] = {0, 0, 1, 1, 1, -1, -1, -1};
    int vis[N][N], lst[N][N], total, mobility[N][N], id[TOTAL], stability[N][N], Tclock;

    double t1, t2, p1, p2, m; //评估值
    double value[TOTAL];
    bool stableFlag;

    node opt[TOTAL][3];

    int sgn(const int &x);
    bool CheckPosition(const int &x, const int &y);
    bool CheckAvailability(const node &s, const node &t);
    bool CheckOperation(const node &s, const node &t, const node &b, const int &id);
    bool ExecuteOperation(const node &s, const node &t, const node &b, const int &id);
    void WithDraw(const node &s, const node &t, const node &b, const int &id);


    //评估函数 1.0  初级评估函数 int
    int Value1(const int &id);
    int Evaluate1();

    //评估函数 2.0 评估值类型为 double
    void check_stable(const int &id);
    void calc_mobility();
    double calcM(const int &id);
    void bfsQ(const int &id);
    void EvaluateQ();
    void bfsK(const int &id);
    void EvaluateK();
    //评估 2.0
    double Evaluate2();

    //决策
    void DecisionMaking1();
    void DecisionMaking2();
    //决策 3.0 双层遍历版本 估价函数稍更改, 双层搜索 + MinMax 优化 + AlphaBeta 剪枝
    //发现在决策很多的时候大概率超时, 解决方案: 在一层的时候评估一下
    bool MinEvaluate(double &MaxValue, const double &extra);
    void DecisionMaking3();

};

#endif // AMAZON_H
```

2.头文件 mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QDialog>
#include "amazon.h"

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private:
    Ui::MainWindow *ui;
    amazon *Amazon;

    int stepCount, operateCount, isAI, isBeing, isRunning, Time;
    void paintEvent(QPaintEvent *event);
    void mousePressEvent(QMouseEvent *event);
    void GameRunning();
    void NewGame(); //运行部分
    void AIplay();
    void Record();
    void LoadDoit();
    void Load();
    void Quit();
};

#endif // MAINWINDOW_H
```

3.源文件 amazon.cpp

```
#include "amazon.h"
#include "mainwindow.h"
#include <cmath>
#include <ctime>
```

```cpp
#include <cstdio>
#include <iostream>
#include <algorithm>

amazon :: amazon () {
    Initialize();
}

void amazon :: Initialize () { //初始化棋盘
    for (int i = 0; i < N; ++i)
        for (int j = 0; j < N; ++j) map[i][j] = 0;

    for (int i = 0; i < N; ++i)
        for (int j = 0; j < N; ++j) expected[i][j] = 0;

    map[0][2] = map[2][0] = map[5][0] = map[7][2] = 1;
    map[0][5] = map[2][7] = map[5][7] = map[7][5] = -1;
    TurnCount = 0, curBotColor = -1;
}

int amazon :: sgn(const int &x) {
    if (!x) return 0;
    return x < 0 ? -1 : 1;
}

bool amazon :: CheckPosition(const int &x, const int &y) {
    if (x < 0 || x >= N) return false;
    if (y < 0 || y >= N) return false;
    return true;
}

//一些基础操作
#define sx s.first
#define sy s.second
#define tx t.first
#define ty t.second
#define bx b.first
#define by b.second

bool amazon :: CheckAvailability(const node &s, const node &t) { //s,t 在棋盘上, s -> t 路径上无棋子或障碍(不包括 s,t 位置)
    if (!CheckPosition(sx, sy) || !CheckPosition(tx, ty)) return false;

    int dx = sgn(tx - sx), dy = sgn(ty - sy), nx = sx, ny = sy;
```

```
    for (int i = 1; i; ++i) {
        nx += dx, ny += dy;
        if (!CheckPosition(nx, ny)) return false;
        if (nx == tx && ny == ty) return true;
        if (map[nx][ny]) return false;
    }

    return false; //这行实际上不会用到，仅仅为了程序的规范性
}

bool amazon :: CheckOperation(const node &s, const node &t, const node &b, const int &id)
{ //判断操作是否合理
    if (!CheckPosition(sx, sy) || map[sx][sy] != id) return false;
    if (!CheckPosition(tx, ty) || map[tx][ty]) return false;
    if (!CheckPosition(bx, by)) return false;
    if (map[bx][by] && (bx != sx || by != sy)) return false;
    map[sx][sy] = 0;//注意起始位置在目标位置和障碍位置连线上的情况
    bool flag = CheckAvailability(s, t) && CheckAvailability(t, b);
    map[sx][sy] = id;
    return flag;
}

bool amazon :: ExecuteOperation(const node &s, const node &t, const node &b, const int &id)
{ //执行一次操作
    if (CheckOperation(s, t, b, id)) return map[tx][ty] = map[sx][sy], map[sx][sy] = 0,
map[bx][by] = 2, true;
    return false;
}

void amazon :: WithDraw(const node &s, const node &t, const node &b, const int &id) {
    map[bx][by] = 0, map[sx][sy] = map[tx][ty], map[tx][ty] = 0;
}

#undef sx
#undef sy
#undef tx
#undef ty
#undef bx
#undef by

//判断执子方能否走步
bool amazon :: Judge() {
    total = 0;
```

```
    for (int i = 0; i < N; ++i)
        for (int j = 0; j < N; ++j) if (map[i][j] == curBotColor) {
            int x = i, y = j;
            for (int k = 0; k < 8; ++k) {
                int nx = x, ny = y;
                for (int ki = 1; ; ++ki) {
                    nx += gox[k], ny += goy[k];
                    if (!CheckPosition(nx, ny) || map[nx][ny]) break;
                    for (int l = 0; l < 8; ++l) {
                        int wx = nx, wy = ny;
                        for (int li = 1; ; ++li) {
                            wx += gox[l], wy += goy[l];
                            if (!CheckPosition(wx, wy)) break;
                            if (map[wx][wy] && (wx != x || wy != y)) break;

                            return true;
                        }
                    }
                }
            }
        }

    return false;
}

//评估函数1.0  初级评估函数 int
int amazon :: Value1(const int &id) {
    for (int i = 0; i < N; ++i)
        for (int j = 0; j < N; ++j) vis[i][j] = 0;

    for (int i = 0; i < N; ++i)
        for (int j = 0; j < N; ++j) if (map[i][j] == id) {
            for (int k = 0; k < 8; ++k) {
                int nx = i, ny = j;
                for (int ki = 1; ki; ++ki) {
                    nx += gox[k], ny += goy[k];
                    if (!CheckPosition(nx, ny)) break;
                    if (map[nx][ny]) break;

                    vis[nx][ny] = 1;
                }
            }
        }
```

```
    int cnt = 0;
    for (int i = 0; i < N; ++i)
        for (int j = 0; j < N; ++j) cnt += vis[i][j];
    return cnt;
}

int amazon :: Evaluate1() {
    return Value1(curBotColor) - Value1(-curBotColor);
}

//评估函数 2.0 评估值类型为 double
void amazon :: check_stable(const int &id) {
    for (int x = 0; x < N; ++x)
        for (int y = 0; y < N; ++y) if (map[x][y] == id) {
            stability[x][y] = true;

            for (int i = 0; i < N; ++i)
                for (int j = 0; j < N; ++j) vis[i][j] = 0;

            while (!que.empty()) que.pop();
            vis[x][y] = 1, que.push(std :: make_pair(x, y));

            while (!que.empty()) {
                node cur = que.front(); que.pop();
                int cx = cur.first, cy = cur.second;
                for (int k = 0; k < 8; ++k) {
                    int nx = cx + gox[k], ny = cy + goy[k];
                    if (!CheckPosition(nx, ny)) continue;

                    if (vis[nx][ny]) continue;
                    vis[nx][ny] = 1;

                    if (map[nx][ny] == -id) stability[x][y] = false;
                    if (map[nx][ny]) continue;

                    que.push(std :: make_pair(nx, ny));
                }
            }
        }

    return;
}

void amazon :: calc_mobility() {
```

```
    for (int i = 0; i < N; ++i)
        for (int j = 0; j < N; ++j) {
            mobility[i][j] = 0;
            for (int k = 0; k < 8; ++k) {
                int nx = i + gox[k], ny = j + goy[k];
                if (!CheckPosition(nx, ny) || map[nx][ny]) continue;
                ++mobility[i][j];
            }
        }

    return;
}

double amazon :: calcM(const int &id) {
    double res = 0, mn = inf;
    for (int i = 0; i < N; ++i)
        for (int j = 0; j < N; ++j) if (map[i][j] == id) {
            for (int k = 0; k < 8; ++k) {
                int nx = i, ny = j;
                for (int ki = 1; ki; ++ki) {
                    nx += gox[k], ny += goy[k];
                    if (!CheckPosition(nx, ny)) break;
                    if (map[nx][ny]) break;

                    res += (double)mobility[nx][ny] / (double)ki;
                    mn = std :: min(mn, (double)mobility[nx][ny] / (double)ki);
                }
            }
        }

    return res + mn;
}

void amazon :: bfsQ(const int &id) {
    while (!que.empty()) que.pop();
    for (int i = 0; i < N; ++i)
        for (int j = 0; j < N; ++j) vis[i][j] = 1e9;

    for (int i = 0; i < N; ++i)
        for (int j = 0; j < N; ++j)
            if (map[i][j] == id)
                vis[i][j] = 0, que.push(std :: make_pair(i, j));

    while (!que.empty()) {
```

```
        node cur = que.front(); que.pop();
        int x = cur.first, y = cur.second;
        for (int k = 0; k < 8; ++k) {
            int nx = x, ny = y;
            for (int ki = 1; ki; ++ki) {
                nx += gox[k], ny += goy[k];
                if (!CheckPosition(nx, ny)) break;
                if (map[nx][ny]) break;
                if (vis[nx][ny] <= vis[x][y] + 1) continue;

                vis[nx][ny] = vis[x][y] + 1;
                que.push(std :: make_pair(nx, ny));
            }
        }
    }

    return;
}

void amazon :: EvaluateQ() {
    t1 = 0, p1 = 0;

    bfsQ(curBotColor);
    for (int i = 0; i < N; ++i)
        for (int j = 0; j < N; ++j) lst[i][j] = vis[i][j];

    bfsQ(-curBotColor);

    stableFlag = true;
    for (int i = 0; i < N; ++i)
        for (int j = 0; j < N; ++j) if (!map[i][j]) {
            if (vis[i][j] == lst[i][j]) t1 += Ec;
            else
                if (vis[i][j] > lst[i][j]) t1 += 1.0;
                else t1 -= 1.0;

            if (vis[i][j] < inf && lst[i][j] < inf) stableFlag = false;

            p1 += 2.0 * (pow(2.0, -lst[i][j]) - pow(2.0, -vis[i][j]));
        }
    return;
}

void amazon :: bfsK(const int &id) {
```

```
    while (!que.empty()) que.pop();
    for (int i = 0; i < N; ++i)
        for (int j = 0; j < N; ++j) vis[i][j] = 1e9;

    for (int i = 0; i < N; ++i)
        for (int j = 0; j < N; ++j)
            if (map[i][j] == id)
                vis[i][j] = 0, que.push(std :: make_pair(i, j));

    while (!que.empty()) {
        node cur = que.front(); que.pop();
        int x = cur.first, y = cur.second;
        for (int k = 0; k < 8; ++k) {
            int nx = x + gox[k], ny = y + goy[k];
            if (!CheckPosition(nx, ny)) continue;
            if (map[nx][ny]) continue;
            if (vis[nx][ny] <= vis[x][y] + 1) continue;

            vis[nx][ny] = vis[x][y] + 1;
            que.push(std :: make_pair(nx, ny));
        }
    }

    return;
}

void amazon :: EvaluateK() {
    t2 = 0, p2 = 0;

    bfsK(curBotColor);
    for (int i = 0; i < N; ++i)
        for (int j = 0; j < N; ++j) lst[i][j] = vis[i][j];

    bfsK(-curBotColor);
    for (int i = 0; i < N; ++i)
        for (int j = 0; j < N; ++j) if (!map[i][j]) {
            if (vis[i][j] == lst[i][j]) t2 += Ec;
            else
                if (vis[i][j] > lst[i][j]) t2 += 1.0;
                else t2 -= 1.0;

            p2 += std :: min(1.0, std :: max(-1.0, ((double)vis[i][j] - lst[i][j]) / 6.0));
        }
    return;
```

```
}

//评估 2.0
double amazon :: Evaluate2() {
    calc_mobility();
    m = (calcM(curBotColor) - calcM(-curBotColor)) / 30;

    EvaluateQ();
    EvaluateK();

    if (stableFlag) return t1;
    if (TurnCount < 10) return 0.14 * t1 + 0.37 * t2 + 0.13 * p1 + 0.13 * p2 + 0.20 * m;
    if (TurnCount < 25) return 0.30 * t1 + 0.25 * t2 + 0.20 * p1 + 0.20 * p2 + 0.05 * m;
    return 0.80 * t1 + 0.10 * t2 + 0.05 * p1 + 0.05 * p2;
}


//决策 1.0 随机版本 目标:能够在规定时间内正常输出的随机 Bot
void amazon :: DecisionMaking1() { //这里枚举了所有可能的下法，以便之后随机
    total = 0;

    for (int i = 0; i < N; ++i)
        for (int j = 0; j < N; ++j) if (map[i][j] == curBotColor) {
            int x = i, y = j;
            for (int k = 0; k < 8; ++k) {
                int nx = x, ny = y;
                for (int ki = 1; ; ++ki) {
                    nx += gox[k], ny += goy[k];
                    if (!CheckPosition(nx, ny) || map[nx][ny]) break;
                    for (int l = 0; l < 8; ++l) {
                        int wx = nx, wy = ny;
                        for (int li = 1; ; ++li) {
                            wx += gox[l], wy += goy[l];
                            if (!CheckPosition(wx, wy)) break;
                            if (map[wx][wy] && (wx != x || wy != y)) break;

                            opt[total][0] = std :: make_pair(x, y);
                            opt[total][1] = std :: make_pair(nx, ny);
                            opt[total][2] = std :: make_pair(wx, wy);
                            ++total;
                        }
                    }
                }
            }
```

```
        }


    if (!total) { //本机操作中, 这种情况不会出现
        for (int i = 0; i < 6; ++i) std :: cout << -1 << ' ';
        putchar('\n');
        return;
    }

    std :: srand(time(NULL));
    int K = rand() % total;

    /*简单交互输出
    for (int i = 0; i < 3; ++i)
        printf("%d %d ", opt[K][i].first, opt[K][i].second);*/

    // 本机操作中不需要输出AI指令 只需记录
    for(int i = 0; i < 3; ++i)
        Num[i * 2] = opt[K][i].first, Num[i * 2 + 1] = opt[K][i].second;

    return;
}


//决策2.0 单层遍历版本 简单估价使它跑过随机
void amazon :: DecisionMaking2() {
    total = 0;

    for (int i = 0; i < N; ++i)
        for (int j = 0; j < N; ++j) if (map[i][j] == curBotColor) {
            int x = i, y = j;
            for (int k = 0; k < 8; ++k) {
                int nx = x, ny = y;
                for (int ki = 1; ; ++ki) {
                    nx += gox[k], ny += goy[k];
                    if (!CheckPosition(nx, ny) || map[nx][ny]) break;
                    for (int l = 0; l < 8; ++l) {
                        int wx = nx, wy = ny;
                        for (int li = 1; ; ++li) {
                            wx += gox[l], wy += goy[l];
                            if (!CheckPosition(wx, wy)) break;
                            if (map[wx][wy] && (wx != x || wy != y)) break;

                            opt[total][0] = std :: make_pair(x, y);
```

```
                        opt[total][1] = std :: make_pair(nx, ny);
                        opt[total][2] = std :: make_pair(wx, wy);
                        ++total;
                    }
                }
            }
        }
    }

    if (!total) { //本机操作中, 这种情况不会出现
        for (int i = 0; i < 6; ++i) std :: cout << -1 << ' ';
        putchar('\n');
        return;
    }

    int MaxValue = -inf, MaxID = -1;
    for (int i = 0; i < total; ++i) {
        ExecuteOperation(opt[i][0], opt[i][1], opt[i][2], curBotColor);
        int res = Evaluate1();
        if (res > MaxValue) MaxValue = res, MaxID = i;
        WithDraw(opt[i][0], opt[i][1], opt[i][2], curBotColor);
    }

    if (~MaxID) {
        /*简单交互输出
        for (int i = 0; i < 3; ++i)
            printf("%d %d ", opt[MaxID][i].first, opt[MaxID][i].second);*/

        // 本机操作中不需要输出 AI 指令 只需记录
        for(int i = 0; i < 3; ++i)
            Num[i * 2] = opt[MaxID][i].first, Num[i * 2 + 1] = opt[MaxID][i].second;
    }
    else { //本机操作中, 这种情况不会出现
        for (int i = 0; i < 6; ++i) std :: cout << -1 << ' ';
        putchar('\n');
    }

    return;
}

//决策 3.0 双层遍历版本 估价函数稍更改, 双层搜索 + MinMax 优化 + AlphaBeta 剪枝
//发现在决策很多的时候大概率超时, 解决方案: 在一层的时候评估一下
bool amazon :: MinEvaluate(double &MaxValue, const double &extra) { //二层搜索
    double mn = inf;
```

```
    for (int i = 0; i < N; ++i)
        for (int j = 0; j < N; ++j) if (map[i][j] == -curBotColor) {
            int x = i, y = j;
            for (int k = 0; k < 8; ++k) {
                int nx = x, ny = y;
                for (int ki = 1; ; ++ki) {
                    nx += gox[k], ny += goy[k];
                    if (!CheckPosition(nx, ny) || map[nx][ny]) break;
                    for (int l = 0; l < 8; ++l) {
                        int wx = nx, wy = ny;
                        for (int li = 1; ; ++li) {
                            wx += gox[l], wy += goy[l];
                            if (!CheckPosition(wx, wy)) break;
                            if (map[wx][wy] && (wx != x || wy != y)) break;

                            ExecuteOperation(std :: make_pair(x, y), std :: make_pair(nx,
ny), std :: make_pair(wx, wy), -curBotColor);
                            double res = Evaluate2() + extra;
                            WithDraw(std :: make_pair(x, y), std :: make_pair(nx, ny),
std :: make_pair(wx, wy), -curBotColor);

                            if (res <= MaxValue) return false;
                            mn = std :: min(res, mn);
                        }
                    }
                }
            }
        }

    if (mn == inf) {
        mn = Evaluate2() + extra + 1e5; //一个胜利的方案应该是优先考虑的方案，同时有多个胜利
方案时考虑权值最大的
        if (mn <= MaxValue) return false;
    }

    MaxValue = mn;

    return true;
}

void amazon :: DecisionMaking3() {
    total = 0;
    Tclock = clock();
```

```
    check_stable(curBotColor);
    for (int i = 0; i < N; ++i)
        for (int j = 0; j < N; ++j) if (map[i][j] == curBotColor) {
            int x = i, y = j;
            for (int k = 0; k < 8; ++k) {
                int nx = x, ny = y;
                for (int ki = 1; ; ++ki) {
                    nx += gox[k], ny += goy[k];
                    if (!CheckPosition(nx, ny) || map[nx][ny]) break;
                    for (int l = 0; l < 8; ++l) {
                        int wx = nx, wy = ny;
                        for (int li = 1; ; ++li) {
                            wx += gox[l], wy += goy[l];
                            if (!CheckPosition(wx, wy)) break;
                            if (map[wx][wy] && (wx != x || wy != y)) break;

                            opt[total][0] = std :: make_pair(x, y);
                            opt[total][1] = std :: make_pair(nx, ny);
                            opt[total][2] = std :: make_pair(wx, wy);

                            ExecuteOperation(opt[total][0], opt[total][1], opt[total][2],
curBotColor);
                            value[total] = Evaluate2() + (stability[i][j] ? 0 : 100),
id[total] = total;
                            WithDraw(opt[total][0], opt[total][1], opt[total][2],
curBotColor);

                            ++total;
                        }
                    }
                }
            }
        }

    if (!total) { //本机操作中, 这种情况不会出现
        for (int i = 0; i < 6; ++i) std :: cout << -1 << ' ';
        putchar('\n');
        return;
    }

    std :: sort(id, id + total, [&](int a, int b){
        return value[a] > value[b];
    });
```

```
    double MaxValue = -inf;
    int MaxID = 0;

    int checkpoint = 50;
    for (int k = 0, i; i = id[k], k < total; ++k) {
        ExecuteOperation(opt[i][0], opt[i][1], opt[i][2], curBotColor);
        if (MinEvaluate(MaxValue, (stability[opt[i][0].first][opt[i][0].second] ? 0 : 100)))
MaxID = i;
        WithDraw(opt[i][0], opt[i][1], opt[i][2], curBotColor);
        if (k == checkpoint) {
            checkpoint += 20;
            if (clock() - Tclock > 500) break;
        }
    }

    if (~MaxID) {
        //简单交互输出
        /*for (int i = 0; i < 3; ++i)
            printf("%d %d ", opt[MaxID][i].first, opt[MaxID][i].second);*/

        //本机操作中不需要输出 AI 指令 只需记录
        for(int i = 0; i < 3; ++i)
            Num[i * 2] = opt[MaxID][i].first, Num[i * 2 + 1] = opt[MaxID][i].second;
    }
    else { //本机操作中, 这种情况不会出现
        for (int i = 0; i < 6; ++i) std :: cout << -1 << ' ';
        putchar('\n');
    }

    return;
}

void amazon :: DecisionMaking() {
    std :: cerr << level << std :: endl; //用于检验 bot 的等级
    if (level == 0) DecisionMaking1();
    else
        if (level == 1) DecisionMaking2();
        else DecisionMaking3();
    return;
}

//记录部分
void amazon :: ReCordInitialize () { //初始化记录文档
```

```
    fp = fopen("record.txt", "w");

    fprintf(fp, "1 %d\n-1 -1 -1 -1 -1 -1\n", level);

    fclose(fp);
    return;
}

void amazon :: ReCordRecord() {
    fp = fopen("record.txt", "r");

    fscanf(fp, "%d%d", &ReCordm, &ReCordlevel); ReCordlen = 0; ReCordflag = false;
    for (int i = 0; i < ReCordm; i++) {
        for (int k = 0; k < 6; ++k) fscanf(fp, "%d", &ReCorda[ReCordlen][k]);
        if(ReCorda[ReCordlen][0] >= 0) ReCordlen++; else ReCordflag = true;

        if (i < ReCordm - 1) {
            for (int k = 0; k < 6; ++k) fscanf(fp, "%d", &ReCorda[ReCordlen][k]);
            ReCordlen++;
        }
    }

    fclose(fp);

    for (int k = 0; k < 6; ++k) ReCorda[ReCordlen][k] = Num[k];
    ++ReCordlen;

    if (!ReCordflag) ++ReCordm;

    fp = fopen("record.txt", "w");

    fprintf(fp, "%d %d\n", ReCordm, ReCordlevel);
    if (!ReCordflag && ReCordm != 1) fprintf(fp, "-1 -1 -1 -1 -1 -1\n");
    for (int i = 0; i < ReCordlen; ++i)
        for (int k = 0; k < 6; ++k) fprintf(fp, "%d%c", ReCorda[i][k], " \n"[k == 5]);

    fclose(fp);

    return;
}

void amazon :: ReCordWithDraw() {
    fp = fopen("record.txt", "r");
```

```
    fscanf(fp, "%d%d", &ReCordm, &ReCordlevel); ReCordlen = 0; ReCordflag = false;
    for (int i = 0; i < ReCordm; i++) {
        for (int k = 0; k < 6; ++k) fscanf(fp, "%d", &ReCorda[ReCordlen][k]);
        if(ReCorda[ReCordlen][0] >= 0) ReCordlen++; else ReCordflag = true;

        if (i < ReCordm - 1) {
            for (int k = 0; k < 6; ++k) fscanf(fp, "%d", &ReCorda[ReCordlen][k]);
            ReCordlen++;
        }
    }

    fclose(fp);

    --ReCordlen;
    if (ReCordflag) --ReCordm;

    fp = fopen("record.txt", "w");

    fprintf(fp, "%d %d\n", ReCordm, ReCordlevel);
    if (!ReCordflag) fprintf(fp, "-1 -1 -1 -1 -1 -1\n");
    for (int i = 0; i < ReCordlen; ++i)
        for (int k = 0; k < 6; ++k) fprintf(fp, "%d%c", ReCorda[i][k], " \n"[k == 5]);

    fclose(fp);

    return;
}

void amazon :: ReCordLoad() { //载入数据 sample 改
    fp = fopen("record.txt", "r");

    if (!fp) { //没有存档文件时, 新建一个空档
        fclose(fp);
        ReCordInitialize (); //初始化记录文档
        fp = fopen("record.txt", "r");
    }

    //读入到当前回合为止，自己和对手的所有行动，从而把局面恢复到当前回合
    fscanf(fp, "%d%d", &TurnCount, &level);
    ReCordm = TurnCount, ReCordlevel = level;

    curBotColor = -1; // 先假设自己是白子
    for (int i = 0; i < TurnCount; i++) {
        int x0, y0, x1, y1, x2, y2;
```

```
        fscanf(fp, "%d%d%d%d%d%d", &x0, &y0, &x1, &y1, &x2, &y2);

        // 首先是对手行动
        if (x0 == -1) curBotColor = 1; //第一回合收到坐标是-1, -1，说明我是黑方
        else ExecuteOperation(std :: make_pair(x0, y0), std :: make_pair(x1, y1), std ::
make_pair(x2, y2), -curBotColor); //模拟对方落子

        // 然后是自己当时的行动
        // 对手行动总比自己行动多一个
        if (i < TurnCount - 1) {
            fscanf(fp, "%d%d%d%d%d%d", &x0, &y0, &x1, &y1, &x2, &y2);
            if (x0 >= 0) ExecuteOperation(std :: make_pair(x0, y0), std :: make_pair(x1,
y1), std :: make_pair(x2, y2), curBotColor); // 模拟己方落子
        }
    }
    if (TurnCount == 0) curBotColor = 1;

    fclose(fp);

    return;
}


void amazon :: expect(const int &x, const int &y, int &stepCount) {
    if (x < 0 || x >= N) return;
    if (y < 0 || y >= N) return;
    if (map[x][y] != curBotColor && expected[x][y] != 1) return;

    if (expected[x][y] != 1) {

        for (int i = 0; i < N; ++i)
            for (int j = 0; j < N; ++j) expected[i][j] = 0;
        expected[x][y] = -1;
        Num[0] = x, Num[1] = y;

        for (int k = 0; k < 8; ++k) {
            int nx = x, ny = y;
            for (int ki = 1; ; ++ki) {
                nx += gox[k], ny += goy[k];
                if (!CheckPosition(nx, ny) || map[nx][ny]) break;
                expected[nx][ny] = 1;
            }
        }
```

```
        stepCount = 1;
    }
    else
        if (stepCount == 1) {

            for (int i = 0; i < N; ++i)
                for (int j = 0; j < N; ++j) expected[i][j] = 0;
            expected[Num[0]][Num[1]] = -1, expected[x][y] = -1;
            Num[2] = x, Num[3] = y;

            map[Num[0]][Num[1]] = 0;
            for (int k = 0; k < 8; ++k) {
                int nx = x, ny = y;
                for (int ki = 1; ; ++ki) {
                    nx += gox[k], ny += goy[k];
                    if (!CheckPosition(nx, ny) || map[nx][ny]) break;
                    expected[nx][ny] = 1;
                }
            }
            map[Num[0]][Num[1]] = curBotColor;

            stepCount = 2;
        }
        else {
            for (int i = 0; i < N; ++i)
                for (int j = 0; j < N; ++j) expected[i][j] = 0;
            Num[4] = x, Num[5] = y;

            stepCount = 3;
        }

    return;
}
```

4.源文件 mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "ui_dialog.h"
#include <QPixmap>
#include <QPainter>
#include <QMainWindow>
#include <QMouseEvent>
#include <iostream>
#include <iomanip>
```

```
#include <QAction>
#include <QPushButton>
#include <QMessageBox>
#include <QDialog>

MainWindow :: MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui :: MainWindow)
{
    ui -> setupUi(this);

//定义各种键
    //主功能键
    connect(ui -> newGameButton, &QPushButton :: clicked, [this] { NewGame(); });
    connect(ui -> quitButton, &QPushButton :: clicked, [this] { Quit(); });
    connect(ui -> recordButton, &QPushButton :: clicked, [this] { Record(); });
    connect(ui -> loadButton, &QPushButton :: clicked, [this] { Load(); });

    //次级键
    connect(ui -> Level0Button, &QPushButton :: clicked, [this] { //"简单"键

        ui -> askingLevelLabel -> setVisible(false);
        ui -> Level0Button -> setVisible(false);
        ui -> Level1Button -> setVisible(false);
        ui -> Level2Button -> setVisible(false);

        Amazon -> level = 0;

        ui -> widget -> raise(); //按键位置重叠, 需要把它的优先级调高
        ui -> askingColorLabel -> setVisible(true); //决定先后手
        ui -> blackButton -> setVisible(true);
        ui -> whiteButton -> setVisible(true);

        return;
    });

    connect(ui -> Level1Button, &QPushButton :: clicked, [this] { //"中等"键

        ui -> askingLevelLabel -> setVisible(false);
        ui -> Level0Button -> setVisible(false);
        ui -> Level1Button -> setVisible(false);
        ui -> Level2Button -> setVisible(false);

        Amazon -> level = 1;
```

```
    ui -> widget -> raise();
    ui -> askingColorLabel -> setVisible(true); //决定先后手
    ui -> blackButton -> setVisible(true);
    ui -> whiteButton -> setVisible(true);

    return;
});

connect(ui -> Level2Button, &QPushButton :: clicked, [this] { //"困难"键

    ui -> askingLevelLabel -> setVisible(false);
    ui -> Level0Button -> setVisible(false);
    ui -> Level1Button -> setVisible(false);
    ui -> Level2Button -> setVisible(false);

    Amazon -> level = 2;

    ui -> widget -> raise();
    ui -> askingColorLabel -> setVisible(true); //决定先后手
    ui -> blackButton -> setVisible(true);
    ui -> whiteButton -> setVisible(true);

    return;
});

connect(ui -> nextButton, &QPushButton :: clicked, [this] { //"对方落子"键 人机对战专用

    if (clock() < Time + 300) {
        Time = clock();
        std :: cerr << Time << std :: endl;
        return;
    };
    Time = clock(); //加时间戳防止在瞬时内多次按键导致连下多步

    if (!isAI) return;

    ui -> nextButton -> setVisible(false);
    ui -> retryButton -> setVisible(false);

    Amazon -> DecisionMaking(); //AI 操作
    Amazon -> ReCordRecord(), ++operateCount;

    ui->recordButton->setVisible(true);
```

```
    Amazon -> Initialize(); //接下来玩家操作
    Amazon -> ReCordLoad();
    repaint();

    if (!Amazon -> Judge()) {
        if (isRunning) {
            QMessageBox :: information(this, tr("游戏结束"), tr("AI 获胜"));

            isRunning = false;
            ui->recordButton->setVisible(false); //游戏结束时不可存档
            ui -> replaceButton -> setVisible(false); //游戏结束时关闭 AI 辅助键
        }

        return;
    }

    isBeing = true, isAI = false; //往后可以鼠标输入
    ui -> replaceButton -> raise();
    ui -> replaceButton -> setVisible(true);

    return;
});

connect(ui -> retryButton, &QPushButton :: clicked, [this] { //"重新落子"键

    if (!isAI) return;
    ui->nextButton->setVisible(false);
    ui->retryButton->setVisible(false);

    Amazon -> ReCordWithDraw(), --operateCount; //悔步(仅供操作失误的情况使用)
    if (!operateCount) ui->recordButton->setVisible(false);

    Amazon -> Initialize(); //接下来玩家重新操作
    Amazon -> ReCordLoad();
    repaint();

    if (!Amazon -> Judge()) {
        if (isRunning) {
            QMessageBox :: information(this, tr("游戏结束"), tr("AI 获胜"));

            isRunning = false;
            ui->recordButton->setVisible(false); //游戏结束时不可存档
            ui -> replaceButton -> setVisible(false); //游戏结束时关闭 AI 辅助键
```

```
        }

        return;
    }

    isBeing = true, isAI = false; //往后可以鼠标输入
    ui -> replaceButton -> raise();
    ui -> replaceButton -> setVisible(true);

    return;
});

connect(ui -> replaceButton, &QPushButton :: clicked, [this] { //"AI 代替落子"键

    if (clock() < Time + 300) {
        Time = clock();
        std :: cerr << Time << std :: endl;
        return;
    };
    Time = clock(); //加时间戳防止在瞬时内多次按键导致连下多步

    if (!isBeing) return;

    Amazon -> Initialize();
    Amazon -> ReCordLoad();
    Amazon -> level = 2; //用最好的 AI 代替玩家落子, 改善游戏体验

    Amazon -> DecisionMaking(); //AI 操作
    Amazon -> ReCordRecord(), ++operateCount;

    ui->recordButton->setVisible(true);

    isBeing = false, isAI = true;
    AIplay(); //接下来 AI 操作

    return;
});

connect(ui -> blackButton, &QPushButton :: clicked, [&] { //"黑"键 (NewGame 专用)
    //It means that "HumanColor = 1"

    ui -> askingColorLabel -> setVisible(false);
    ui -> blackButton -> setVisible(false);
    ui -> whiteButton -> setVisible(false);
```

```
    isRunning = true;

    Amazon -> ReCordInitialize(); //初始化棋局, 确定游戏难度

    Amazon -> Initialize(); //接下来玩家操作
    Amazon -> ReCordLoad();
    repaint();

    isBeing = true, isAI = false; //开启鼠标输入
    ui -> replaceButton -> raise();
    ui -> replaceButton -> setVisible(true);

    return;
});

connect(ui -> whiteButton, &QPushButton :: clicked, [&] { //"白"键 (NewGame 专用)
    //It means that "HumanColor = -1"

    ui -> askingColorLabel -> setVisible(false);
    ui -> blackButton -> setVisible(false);
    ui -> whiteButton -> setVisible(false);

    isRunning = true;

    Amazon -> ReCordInitialize(); //初始化棋局, 确定游戏难度

    isAI = true, isBeing = false; //AI 操作
    AIplay();

    return;
});

connect(ui -> blackButton2, &QPushButton :: clicked, [&] { //"黑"键 (Load 专用)
    //It means that "HumanColor = 1"

    isRunning = true;

    ui -> showingLabel -> setVisible(false);
    ui -> askingColorLabel2 -> setVisible(false);
    ui -> blackButton2 -> setVisible(false);
    ui -> whiteButton2 -> setVisible(false);

    if (Amazon -> curBotColor == 1) { //玩家先手, 这里需要判断是否可以继续游戏
```

```
        Amazon -> Initialize(); //玩家操作
        Amazon -> ReCordLoad();
        repaint();

        if (!Amazon -> Judge()) { //游戏结束
            if (isRunning) {
                QMessageBox :: information(this, tr("消息框"), tr("游戏结束, AI 获胜"));

                isRunning = false;
                ui->recordButton->setVisible(false); //游戏结束时不可存档
                ui -> replaceButton -> setVisible(false); //游戏结束时关闭 AI 辅助键
            }

            return;
        }

        isBeing = true, isAI = false; //接下来开启鼠标操作
        ui -> replaceButton -> raise();
        ui -> replaceButton -> setVisible(true);
    }
    else { //AI 先手
        isAI = true, isBeing = false;
        AIplay(); //AI 操作
    }

    return;
});

connect(ui -> whiteButton2, &QPushButton :: clicked, [&] {
    //It means that "HumanColor = -1"

     isRunning = true;

     ui -> showingLabel -> setVisible(false);
     ui -> askingColorLabel2 -> setVisible(false);
     ui -> blackButton2 -> setVisible(false);
     ui -> whiteButton2 -> setVisible(false);

     if (Amazon -> curBotColor == -1) { //玩家先手, 这里需要判断是否可以继续游戏

        Amazon -> Initialize(); //玩家操作
        Amazon -> ReCordLoad();
        repaint();
```

```
            if (!Amazon -> Judge()) { //游戏结束
                if (isRunning) {
                    QMessageBox :: information(this, tr("游戏结束"), tr("AI 获胜"));

                    isRunning = false;
                    ui->recordButton->setVisible(false); //游戏结束时不可存档
                    ui -> replaceButton -> setVisible(false); //游戏结束时关闭 AI 辅助键
                }
                return;
            }

            isBeing = true, isAI = false; //接下来开启鼠标操作
            ui -> replaceButton -> raise();
            ui -> replaceButton -> setVisible(true);
        }
        else { //AI 先手
            isAI = true, isBeing = false;
            AIplay(); //AI 操作
        }

        return;
    });

    Amazon = new amazon();
    GameRunning();
}

MainWindow :: ~MainWindow()
{
    delete ui;
}

void MainWindow :: paintEvent(QPaintEvent *event) {

    static QPixmap table(":/pic/table.gif");
    static QPixmap arrow(":/pic/arrow.gif");
    static QPixmap goal(":/pic/goal.gif");
    static QPixmap black0(":/pic/black.gif");
    static QPixmap white0(":/pic/white.gif");
    static QPixmap empty0(":/pic/empty.gif");
    static QPixmap black1(":/pic/blackChosen.gif");
    static QPixmap white1(":/pic/whiteChosen.gif");
    static QPixmap empty1(":/pic/emptyChosen.gif");
```

```
    QPainter painter(this);
    painter.drawPixmap(QRectF(0, 0, 600, 600), table, table.rect());

    //str(25, 23) size 70 * 70
    for (int i = 0; i < 8; ++i)
        for (int j = 0; j < 8; ++j) {
            QRectF rec = QRectF(25 + i * 69.5, 23 + j * 70, 70, 70);

            if (!Amazon -> expected[i][j]) {
                if (Amazon -> map[i][j] == 0) {painter.drawPixmap(rec, empty0,
empty0.rect()); continue; }
                if (Amazon -> map[i][j] == 1) {painter.drawPixmap(rec, black0,
empty0.rect()); continue; }
                if (Amazon -> map[i][j] == -1) {painter.drawPixmap(rec, white0,
empty0.rect()); continue; }
                if (Amazon -> map[i][j] == 2) {painter.drawPixmap(rec, arrow,
empty0.rect()); continue; }
            }

            if (Amazon -> expected[i][j] == 1) {painter.drawPixmap(rec, goal, goal.rect());
continue; }

            //Amazon -> expected[i][j] == -1
            if (Amazon -> map[i][j] == 0) {painter.drawPixmap(rec, empty1, empty1.rect());
continue; }
            if (Amazon -> map[i][j] == 1) {painter.drawPixmap(rec, black1, empty1.rect());
continue; }
            if (Amazon -> map[i][j] == -1) {painter.drawPixmap(rec, white1, empty1.rect());
continue; }
        }

    return;
}

void MainWindow :: mousePressEvent(QMouseEvent *event) {
    if (!isBeing) return;

    qreal x = event -> x();
    qreal y = event -> y();
    int posX = ((double)x - 25) / 69.5, posY = ((double)y - 23) / 70; //点定位

    Amazon -> expect(posX, posY, stepCount);
```

```
    repaint(); //重绘界面
    if (stepCount == 3) { //玩家完成决策
        stepCount = 0;
        for (int i = 0; i < amazon :: N; ++i)
            for (int j = 0; j < amazon :: N; ++j) Amazon -> expected[i][j] = 0;

        Amazon -> ReCordRecord(), ++operateCount;
        ui -> recordButton -> setVisible(true);

        isBeing = false, isAI = true;
        AIplay(); //接下来 AI 操作
    }

    return;
}

void MainWindow :: AIplay() {//AI 操作

    Amazon -> Initialize();
    Amazon -> ReCordLoad(); //载入棋局
    repaint();

    if (!Amazon -> Judge()) { //判断终局
        if (isRunning) {
            QMessageBox :: information(this, tr("消息框"), tr("游戏结束, 玩家获胜"));

            isRunning = false;
            ui -> recordButton -> setVisible(false); //游戏结束时不可存档
            ui -> replaceButton -> setVisible(false); //游戏结束时 AI 辅助键关掉
        }

        return;
    }

    ui -> nextButton -> raise();
    ui -> nextButton -> setVisible(true);

    if (operateCount) ui -> retryButton -> setVisible(true);
    //玩家可以选择让对方落子或重新落子

    return;
}
```

```
void MainWindow :: NewGame() { //新游戏

    switch(QMessageBox :: warning(this,tr("消息框"),
           tr("您将开启新游戏"),
           QMessageBox :: Ok|QMessageBox::Cancel,
           QMessageBox :: Ok))
        {
        case QMessageBox :: Ok:
           //次级键/标签要消失
           ui -> nextButton -> setVisible(false);
           ui -> retryButton -> setVisible(false);
           ui -> replaceButton -> setVisible(false);
           ui -> blackButton -> setVisible(false);
           ui -> whiteButton -> setVisible(false);
           ui -> askingColorLabel -> setVisible(false);
           ui -> showingLabel -> setVisible(false);
           ui -> blackButton2 -> setVisible(false);
           ui -> whiteButton2 -> setVisible(false);
           ui -> askingColorLabel2 -> setVisible(false);
           ui -> askingLevelLabel -> setVisible(false);
           ui -> Level0Button -> setVisible(false);
           ui -> Level1Button -> setVisible(false);
           ui -> Level2Button -> setVisible(false);
           ui -> replaceButton -> setVisible(false);

           stepCount = operateCount = 0;
           isAI = isBeing = isRunning = false;

           ui -> recordButton -> setVisible(false);

           Amazon -> Initialize();
           repaint();

           ui -> widget_3 -> raise();
           ui -> askingLevelLabel -> setVisible(true);
           ui -> Level0Button -> setVisible(true);
           ui -> Level1Button -> setVisible(true);
           ui -> Level2Button -> setVisible(true);

          break;
        case QMessageBox :: Cancel:
          break;
        default:
          break;
```

```
        }

    return;
}

void MainWindow :: Record() { //存档

    switch(QMessageBox :: warning(this,tr("消息框"),
            tr("是否存档？"),
            QMessageBox :: Ok|QMessageBox::Cancel,
            QMessageBox :: Ok))
        {
        case QMessageBox :: Ok:
           system("copy record.txt file.txt");
           break;
        case QMessageBox :: Cancel:
           break;
        default:
           break;
        }

    return;
}

void MainWindow :: LoadDoit() { //读档
    FILE* fp = fopen("file.txt", "r");
    fclose(fp);

    if (!fp) { //如果读不到档案, 返回
        QMessageBox :: information(this, tr("消息框"), tr("未读取到存档"));
        return;
    }

    //读档游戏

    //次级键/标签要消失
    ui -> nextButton -> setVisible(false);
    ui -> retryButton -> setVisible(false);
    ui -> replaceButton -> setVisible(false);
    ui -> blackButton -> setVisible(false);
    ui -> whiteButton -> setVisible(false);
    ui -> askingColorLabel -> setVisible(false);
    ui -> showingLabel -> setVisible(false);
    ui -> blackButton2 -> setVisible(false);
```

```
    ui -> whiteButton2 -> setVisible(false);
    ui -> askingColorLabel2 -> setVisible(false);
    ui -> askingLevelLabel -> setVisible(false);
    ui -> Level0Button -> setVisible(false);
    ui -> Level1Button -> setVisible(false);
    ui -> Level2Button -> setVisible(false);

    system("copy file.txt record.txt"); //读档

    stepCount = operateCount = 0;
    isAI = isBeing = isRunning = false;
    ui -> recordButton -> setVisible(false); //新读档的棋局不需要存档

    Amazon -> Initialize();
    Amazon -> ReCordLoad(); //加载棋局
    repaint();

    //接下来决定先后手
    if (Amazon -> curBotColor == 1) ui -> showingLabel -> setText("当前黑方执子");
    else  ui -> showingLabel -> setText("当前白方执子");
    ui -> showingLabel -> setVisible(true);
    ui -> askingColorLabel2 -> setVisible(true);
    ui -> blackButton2 -> setVisible(true);
    ui -> whiteButton2 -> setVisible(true);

    isAI = isBeing = false;

    return;
}

void MainWindow :: Load() { //读档

    switch(QMessageBox :: warning(this,tr("消息框"),
            tr("您将读档。"),
            QMessageBox :: Ok|QMessageBox::Cancel,
            QMessageBox :: Ok))
       {
       case QMessageBox :: Ok:
           LoadDoit();
           break;
       case QMessageBox::Cancel:
           return;
           break;
       default:
```

```
            break;
        }

    return;
}

void MainWindow :: Quit() { //退出

    if (isRunning && operateCount) { //如果棋局未到终局且玩家或 AI 操作过
        switch(QMessageBox :: warning(this,tr("消息框"),
                tr("当前棋局可能未保存, 是否存档? "),
                QMessageBox :: Save|QMessageBox::Discard|QMessageBox::Cancel,
                QMessageBox :: Save))
            {
            case QMessageBox :: Save:
                system("copy record.txt file.txt");
                close();
                break;
            case QMessageBox :: Discard:
                close();
                break;
            case QMessageBox :: Cancel:
                return;
                break;
            default:
                break;
            }
    }
    else {
        switch(QMessageBox :: warning(this,tr("消息框"),
                tr("您将退出游戏桌。"),
                QMessageBox :: Ok|QMessageBox::Cancel,
                QMessageBox :: Ok))
            {
            case QMessageBox :: Ok:
                close();
                break;
            case QMessageBox :: Cancel:
                return;
                break;
            default:
                break;
            }
    }
```

```
    return;
}

void MainWindow :: GameRunning() { //初始化

    stepCount = operateCount = 0;
    isAI = isBeing = isRunning = false;

    //功能键
    ui -> newGameButton -> setVisible(true);
    ui -> quitButton -> setVisible(true);
    ui -> recordButton -> setVisible(false); //初始时存档键不显示
    ui -> loadButton -> setVisible(true);

    //次级键/标签要消失
    ui -> nextButton -> setVisible(false);
    ui -> retryButton -> setVisible(false);
    ui -> replaceButton -> setVisible(false);
    ui -> blackButton -> setVisible(false);
    ui -> whiteButton -> setVisible(false);
    ui -> askingColorLabel -> setVisible(false);
    ui -> showingLabel -> setVisible(false);
    ui -> blackButton2 -> setVisible(false);
    ui -> whiteButton2 -> setVisible(false);
    ui -> askingColorLabel2 -> setVisible(false);
    ui -> askingLevelLabel -> setVisible(false);
    ui -> Level0Button -> setVisible(false);
    ui -> Level1Button -> setVisible(false);
    ui -> Level2Button -> setVisible(false);

    return;
}
```

5.源文件 main.cpp

```
#include "mainwindow.h"
#include <QApplication>
#include <QWidget>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);

    MainWindow w;
```

```
    w.setFixedSize(800, 600);
    w.setWindowTitle("Amazon");

    w.show();

    return a.exec();
}
```