哈爾濱工業大學（威海）

Harbin Institute of Technology at Weihai

程序设计实践报告

题　　目：C++ 中官象棋规则设计与实现

院　　系：计算机科学与技术暨软件学院

角色	学号	姓名	组内评分	小组成绩
组长	**160720211**	卢奕阳	**A**	
组员	**160720126**	李逸霆	**B**	
组员	**160710323**	田世先	**B**	
组员				

哈尔滨工业大学（威海）

二零一九年七月

哈尔滨工业大学（威海）程序设计实践任务书

姓　　名：卢奕阳

学　　号：160720211

院（系）：　计算机科学与技术暨软件学院　　专　　业：　　软件工程

任务起止日期：2019 年 6 月 17 日 至　2019 年 7 月 5 日

课程设计题目：C++ 中宫象棋规则设计与实现

系统可行性分析及主要功能：

1.题目背景：

中国象棋是起源于中国的一种棋戏，在中国有着悠久的历史。由于用具简单，趣味性强，成为流行极为广泛的棋艺活动。已有几千年历史、充满东方智慧的中国象棋在中国的群众基础远超围棋，一直是普及最广的棋类项目。

随着中国象棋的推广也出现了非常多的变体，其他国家的象棋比如朝鲜象棋有许多都是从中国象棋中演变而来，不同的象棋变体使得象棋的趣味性益发增强，让人不会厌倦，同时还发挥了广大人民的创造性，使得游戏不再必须按照既定的规则来，自己创造自己的规则。

中宫象棋就是中国象棋的一种变体，是在中华文化的传统精粹上加以变革，形成更具趣味性的电脑游戏，以供用户娱乐，同时加深对象棋这种传统棋戏的了解。另外，中宫象棋有助于促进对个体智力的开发和训练，可进一步扩展思维的空间，有助于培养想象力、创造性以及思维的灵活性、敏捷性和求异性等。

2.实现意义：

象棋游戏在网络平台上数不胜数，但是针对传统象棋进行改革更新进而开发出其他更具趣味性的游戏却少之又少。因此，我们打算来做这件事，开发一款另类象棋游戏--中宫象棋，以供提高象棋爱好者们的体验。

3.主要问题：

中宫象棋规则：

1、棋具为中国象棋棋子完整一副，棋盘 9x9 线格正方形，没有河界，双 方也没有各自的九宫。

2、只有一个共用的中心九宫，只有将帅受九宫限制，不能出宫。

3、棋子的基本走法吃法同中国象棋，只是士、象可走全盘有效位置、兵 卒起步就可以横移走或吃，重要的是这里将帅没有"飞将"规则，是可以 照面的。

4、虽然取胜方法还是吃掉对方的将或帅，但是一定要"白吃"才算胜利， 即：吃掉敌方将或帅的棋不能随即就被敌方吃掉，因为吃掉对方刚吃掉 将或帅的棋也为胜。就是说，当自己的棋看住自己的将帅时，敌方是不 能吃的，此情况下叫杀将帅可称之为"牵将"，不属于"将军"，被"牵将"时 可以不应将，但此时不能全部撤离被"牵将"保护棋。只有在要杀无保护 的将帅时，才算"将军"，解将的方式多了一项"看护"，此时若无法解将 为输棋。

5、围困敌将或帅四面皆为死路敌方又无法随即打通时也为胜：将或帅周 围横竖四个紧邻点不能没有空位，该空位称作"气点"，"气点"不限中宫 内外，将帅即使不动也必须有自己的"气点"才行，无论这个气点能不能 走，被围堵无气点的将或帅为被"困将"，被困方必须随即解困，无法解 困时为输。不能主动给自己的将或帅造成无"气点"局面，自困为输。

6、中宫象棋关于兵（卒）进到对方底线的规则：1、五枚底线兵为胜。2、 四枚底线兵可兑掉敌人一枚车。3、三枚底线兵可对掉敌人一匹马。4、 两枚底线兵可对掉敌人一枚炮。5、一枚底线兵可兑掉敌人一枚卒。兑掉 等同走棋。

4.主要功能：

走子功能；
牵将功能；
兑子功能
胜负判定功能；
悔棋功能；
重来功能；
退出游戏功能；

工作量：

准备资料：2-3 天
实现象棋规则：2-3 天
人人对战：1-2 天
悔棋重来功能：3 天以上
项目优化与收尾：1-2 天

工作计划安排：

1. 查找资料，为代码编写实现功能做准备。
2. 第一步实现人与人对战的下棋界面，实现能够下棋。
3. 第二步实现悔棋功能。
4. 第三步在前面的基础上实现重新开始及退出游戏功能。
5. 如果还有时间可以对界面进行美化，使得界面更吸引人。

同组设计者及分工：

组长：卢奕阳 学号：160720211 班级：1611201
分工：查找资料，建立数据结构，实现中宫象棋游戏基本逻辑。
组员：田世先 学号：160710323 班级：1611201
分工：悔棋功能，重来退出功能。
组员：李逸霆 学号：160720126 班级：1611201
分工：人人对战，实现界面及美化界面，相关文档书写。

目录

第 1 章 项目简介

1.1 项目背景和意义

项目背景：

中国象棋是起源于中国的一种棋戏，在中国有着悠久的历史。由于用具简单，趣味性强，成为流行极为广泛的棋艺活动。已有几千年历史、充满东方智慧的中国象棋在中国的群众基础远超围棋，一直是普及最广的棋类项目。

随着中国象棋的推广也出现了非常多的变体，其他国家的象棋比如朝鲜象棋有许多都是从中国象棋中演变而来，不同的象棋变体使得象棋的趣味性益发增强，让人不会厌倦，同时还发挥了广大人民的创造性，使得游戏不再必须按照既定的规则来，自己创造自己的规则。

中宫象棋就是中国象棋的一种变体，是在中华文化的传统精粹上加以变革，形成更具趣味性的电脑游戏，以供用户娱乐，同时加深对象棋这种传统棋戏的了解。另外，中宫象棋有助于促进对个体智力的开发和训练，可进一步扩展思维的空间，有助于培养想象力、创造性以及思维的灵活性、敏捷性和求异性等。

实现意义：

象棋游戏在网络平台上数不胜数，但是针对传统象棋进行改革更新进而开发出其他更具趣味性的游戏却少之又少。因此，我们打算来做这件事，开发一款另类象棋游戏--中宫象棋，以供提高象棋爱好者们的体验。

1.2 项目主要研究内容

项目主要实现了中宫象棋的特殊规则：

1、棋具为中国象棋棋子完整一副，棋盘 9x9 线格正方形，没有河界，双方也没有各自的九宫。

2、只有一个共用的中心九宫，只有将帅受九宫限制，不能出宫。

3、棋子的基本走法吃法同中国象棋，只是士、象可走全盘有效位置、兵卒起步就可以横移走或吃，重要的是这里将帅没有"飞将"规则，是可以照面的。

4、虽然取胜方法还是吃掉对方的将或帅，但是一定要"白吃"才算胜利，即：

吃掉敌方将或帅的棋不能随即就被敌方吃掉，因为吃掉对方刚吃掉将或帅的棋也为胜。就是说，当自己的棋看住自己的将帅时，敌方是不能吃的，此情况下叫杀将帅可称之为"牵将"，不属于"将军"，被"牵将"时可以不应将，但此时不能全部撤离被"牵将"保护棋。只有在要杀无保护的将帅时，才算"将军"，解将的方式多了一项"看护"，此时若无法解将为输棋。

5、围困敌将或帅四面皆为死路敌方又无法随即打通时也为胜：将或帅周围横竖四个紧邻点不能没有空位，该空位称作"气点"，"气点"不限中宫内外，将帅即使不动也必须有自己的"气点"才行，无论这个气点能不能走，被围堵无气点的将或帅为被"困将"，被困方必须随即解困，无法解困时为输。不能主动给自己的将或帅造成无"气点"局面，自困为输。

6、中宫象棋关于兵（卒）进到对方底线的规则：1、五枚底线兵为胜；2、四枚底线兵可兑掉敌人一枚车；3、三枚底线兵可对掉敌人一匹马；4、两枚底线兵可对掉敌人一枚炮；5、一枚底线兵可兑掉敌人一枚卒，兑掉等同走棋。

1.3 项目关键技术介绍

编程工具：

本项目采用 C++ 语言图形界面编程工具 Qt，Qt 是一个由 Qt Company 开发的跨平台 C++ 图形用户界面应用程序开发框架。它既可以开发 GUI 程序，也可用于开发非 GUI 程序，比如控制台工具和服务器。Qt 是面向对象的框架，使用特殊的代码生成扩展以及一些宏，Qt 很容易扩展，并且允许真正地组件编程。

平台：

由于采用 Qt 5.7.0 编程工具在 Windows 上编程，因此可以很方便地生成 Windows 版及 Android 版程序，一次开发多次生成，以便在多平台上进行移植运行。

第 2 章 项目设计

2.1 项目模块划分及分工

系统功能模块划分如图 2-1，本项目主要分为三大功能模块，包括人人对战、人机对战及游戏控制。人人对战中实现了棋盘界面展示、棋子按规则移动、牵将保护和气的判定和底线兵兑子三大特殊规则，以及最重要的胜负判定功能。人机对战在人人对战的基础上进行实现，兑子功能较为复杂，因此在人机对战模块中电脑不会使用兑子而用户可以，其他基本规则与人人对战相同，主要增加了局势判断和自动下棋两大核心功能，这两个功能的实现保证了人机对战的可玩性。另外，悔棋功能、重来功能、退出游戏功能属于象棋游戏的基本控制，也进行了实现。

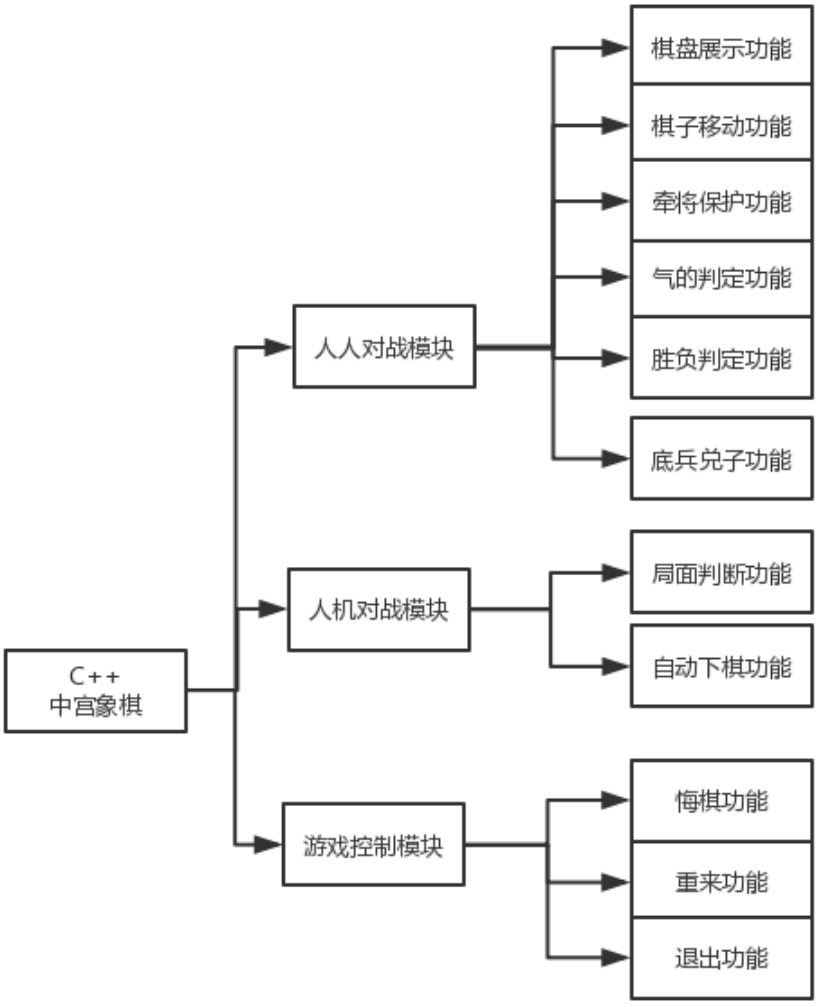

图 2-1 系统功能模块图

具体的分工是卢奕阳负责总体结构设计及人机对战的算法实现，李逸霆负责人人对战的几个基本规则的具体实现，田世先负责游戏控制模块包括悔棋、重来、退出及界面绘制的实现，最后由卢奕阳整合代码，李逸霆负责整理文档。

2.2 总体模块功能及设计

系统总体功能流程如图 2-2，用户初次打开应用程序时会弹出选项，选择人人对战或人机对战，如果选择人人对战则玩家与玩家单机进行对局，若选择人机对战则玩家与电脑进行对局，当胜负判定功能判断出现胜负时结束游戏。游戏过程中也可以直接点击退出按钮退出游戏。

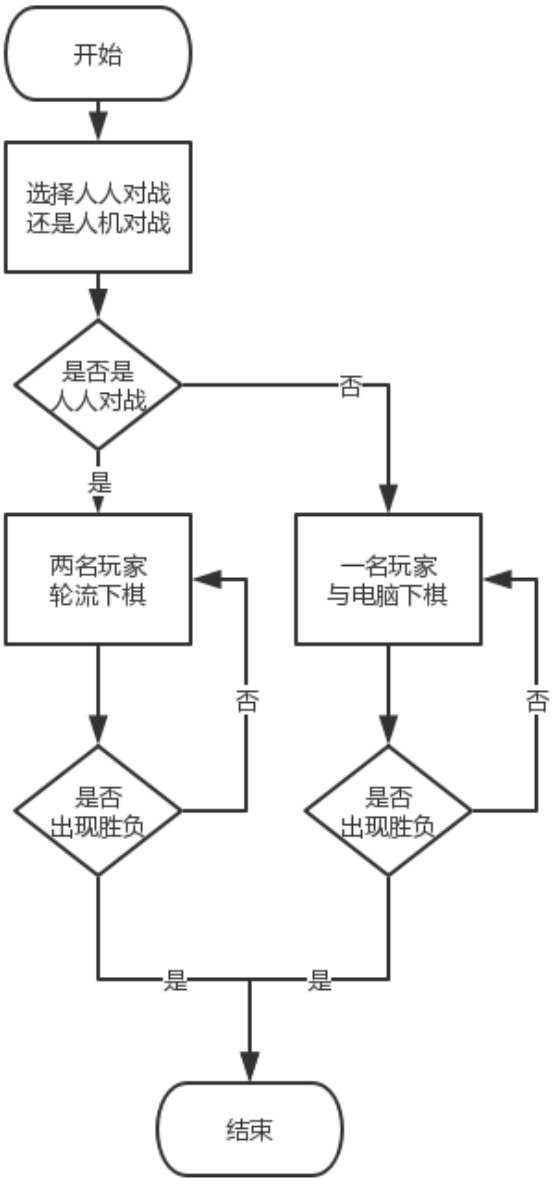

图 2-2 总体流程图

系统类图如图 2-3，一共有 6 个类。

其中 Stone 类即棋子类，规定了棋子在程序中的表示形式。

Step 类即棋步类，包含了移动棋子过程中的起点终点及移动的棋子 id 和被吃的棋子 id，其中棋盘上空的位置默认为棋子 id 是-1。

Board 类即棋盘类，实例化了多个棋子，包含了棋盘绘制及人人对战中棋子移动的规则，通过对 Step 棋步的记录实现了悔棋等功能。

SingleGame 类继承自 Board 类，在人人对战的基础上实现了人机对战中电脑的下棋算法。实现的算法是一个剪枝后的 5 层的最大值最小值算法，超过 5 层的话电脑运行起来会非常卡顿，影响游戏体验，也因此这个算法并没有非常智能，还是会经常走奇怪的棋步。

Dialog 类和 main 类是程序运行开始时的对战模式选择和界面初始化类。

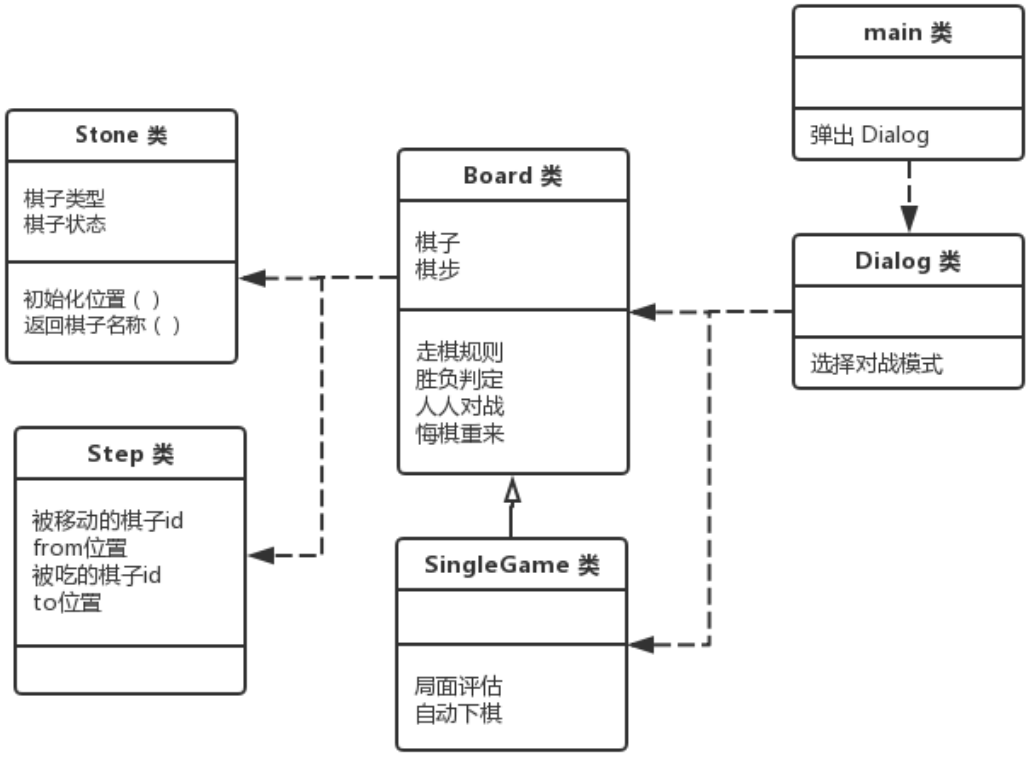

图 2-3 系统类图

第 3 章 项目实现与测试

3.1 人人对战功能模块实现

人人对战功能模块中的胜负判定功能流程图如图 3-1-1 所示：

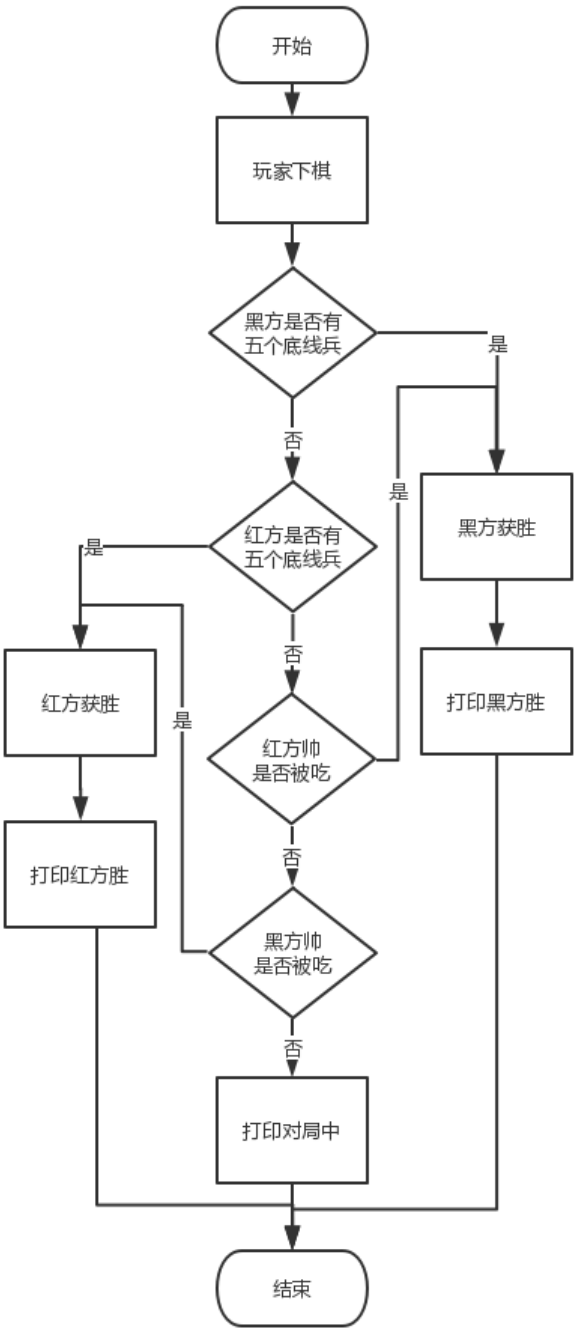

图 3-1-1： 中宫象棋人人对战胜负判定功能流程图

人人对战功能模块中的胜负判定功能核心代码如图 3-1-2 所示：

```
void Board::state()
{
    if(num_of_DXB(!_bRedTurn)==5&&_bRedTurn)
    {
        _state=HEI;
    }
    else if(num_of_DXB(!_bRedTurn)==5&&!_bRedTurn)
    {
        _state=HONG;
    }

    if(_s[15]._dead)_state=HEI;
    else if(_s[31]._dead)_state=HONG;
}

QString Board::getState()
{
    switch(_state)
    {
        case PLAYING:
            return QString::fromLocal8Bit("对局中" );
        case HE:
            return QString::fromLocal8Bit("和棋");
        case HONG:
            return QString::fromLocal8Bit("红方胜");
        case HEI:
            return QString::fromLocal8Bit("黑方胜");
    }
    return QString::fromLocal8Bit("对局状态错误");
}
```

图 3-1-2 胜负状态判定核心代码

人人对战功能模块中的胜负判定功能主要界面如图 3-1-2 所示：

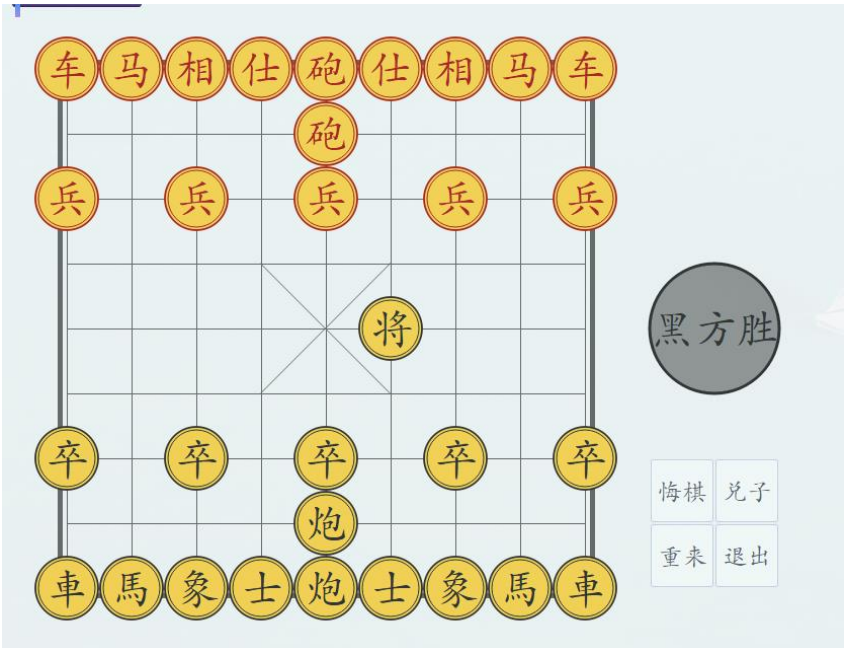

图 3-1-2 胜负状态判定主要界面

3.2 棋子移动功能模块实现

人人对战功能模块中的棋子移动功能流程图如图 3-2-1 所示：

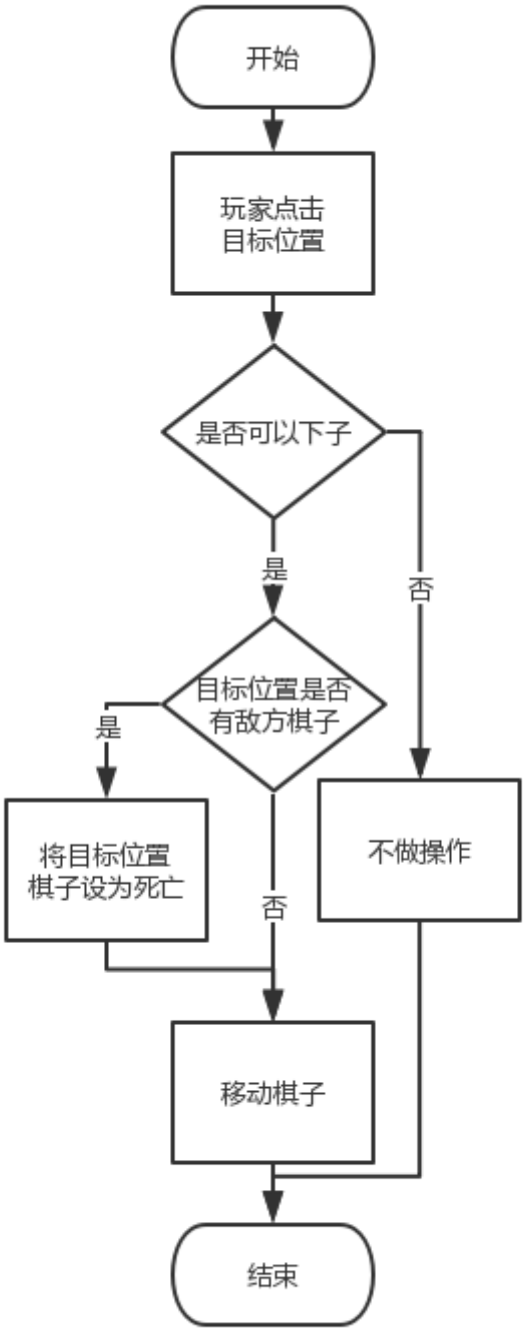

图 3-2-1： 中宫象棋人人对战棋子移动功能流程图

人人对战功能模块中的棋子移动功能核心代码如图 3-2-2 及图 3-2-3 所示：

```
873         if(canMove(_selectid,row,col,id))
874         {
875             moveto(_selectid,row,col,id);
876         }
877
878     }
879 }
880
881 void Board::moveto(int moveid,int row,int col,int killid){
882     //走之前先入栈
883     for(int i=0;i<32;i++)
884     {
885
886         stone[i].push(_s[i]);
887     }
888     /*走棋*/
889     _s[moveid]._row=row;
890     _s[moveid]._col=col;
891
892     if(killid!=-1)
893     {
894         _s[killid]._dead=true;
895         //棋子死后行列要置到棋盘外!!!
896         _s[killid]._row=100;
897         _s[killid]._col=100;
898     }
899
900     //判断是否自己断气
901     qi();
902
903     _s[_selectid]._select=false;
904     _selectid=-1;
905     _bRedTurn=!_bRedTurn;
906     update();
907 }
```

图 3-2-2：棋子移动核心代码 1

```
590 bool Board::canMove(int moveid, int row, int col, int killid)
591 {
592     if(killid==-1||!_s[moveid]._red==_s[killid]._red)
593     {
594         if(canMove0(moveid,row,col,killid))
595         {
596             switch(_s[moveid]._type)
597             {
598             case Stone::JIANG:
599                 return canMove1(moveid,row,col,killid);
600             case Stone::SHI:
601                 return canMove2(moveid,row,col,killid);
602             case Stone::XIANG:
603                 return canMove3(moveid,row,col,killid);
604             case Stone::CHE:
605                 return canMove4(moveid,row,col,killid);
606             case Stone::MA:
607                 return canMove5(moveid,row,col,killid);
608             case Stone::PAO:
609                 return canMove6(moveid,row,col,killid);
610             case Stone::BING:
611                 return canMove7(moveid,row,col,killid);
612             default: break;
613             }
614         }
615         else
616             return false;
617     }
```

图 3-2-3：棋子移动核心代码 2

人人对战功能模块中的棋子移动功能主要界面如图 3-2-4 及 3-2-5 所示：

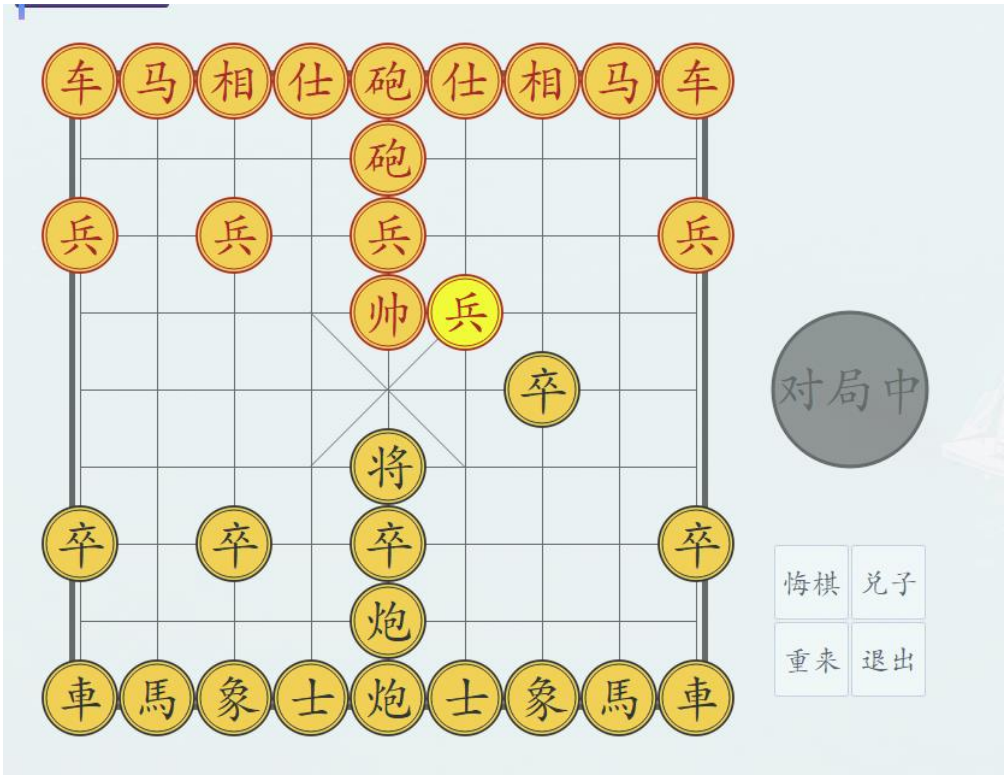

图 3-2-4：棋子移动前界面

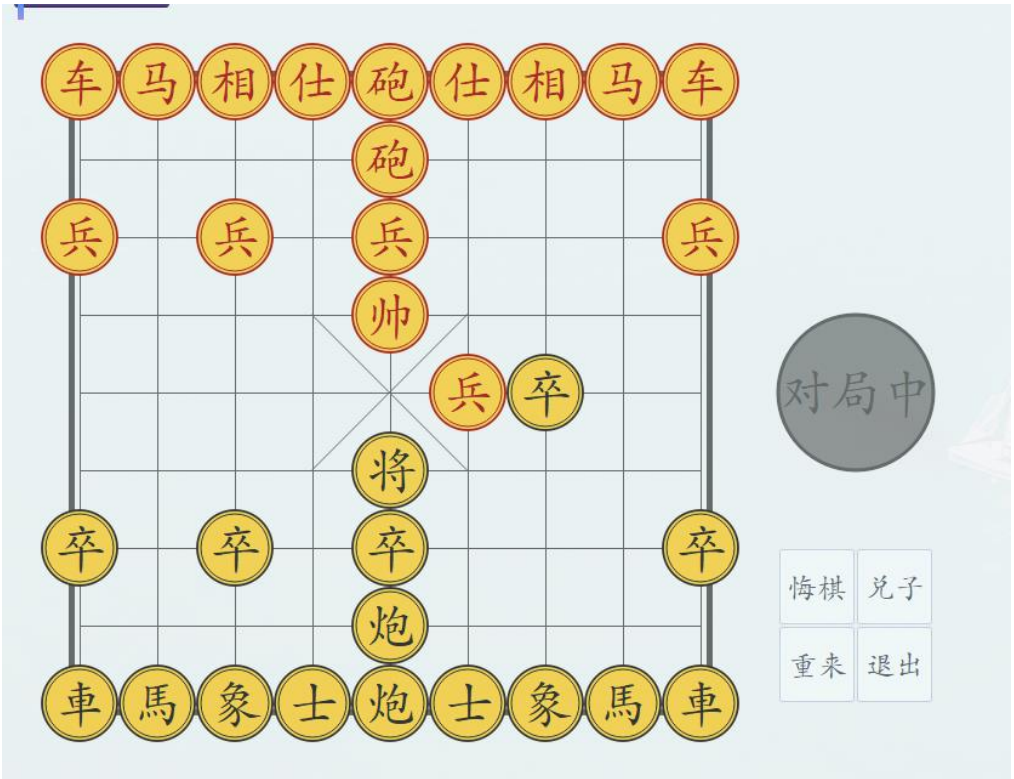

图 3-2-5：棋子移动后界面

3.3 棋子悔棋功能模块实现

游戏控制功能模块中的悔棋功能流程图如图 3-3-1 所示：

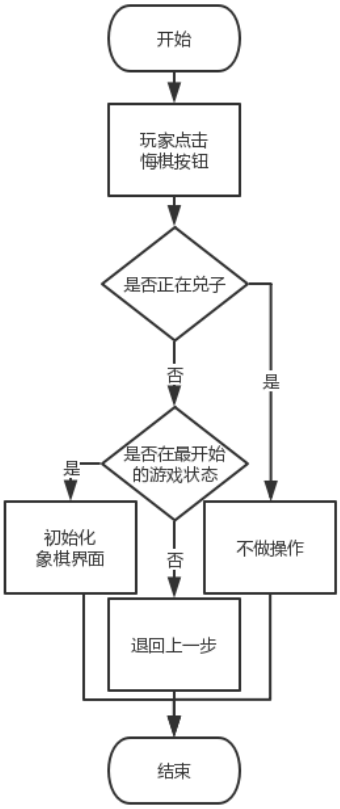

图 3-3-1： 中宫象棋悔棋功能流程图

悔棋功能核心代码如图 3-3-2 所示：

```
1006  void Board::backMove()
1007  {
1008      if(_dz)
1009      {
1010          return;
1011      }
1012      else if(stone->length()==0)
1013      {
1014          go();
1015      }
1016      else if(stone->length()>0)
1017      {
1018          _selectid=-1;
1019          for(int i=0;i<32;i++)
1020          {
1021              _s[i]=stone[i].top();
1022              _s[i]._select=false;
1023              stone[i].pop();
1024          }
1025      }
1026
1027  }
```

图 3-3-2：悔棋功能核心代码

悔棋功能模块中的主要界面如图 3-3-3 及 3-3-4 所示：

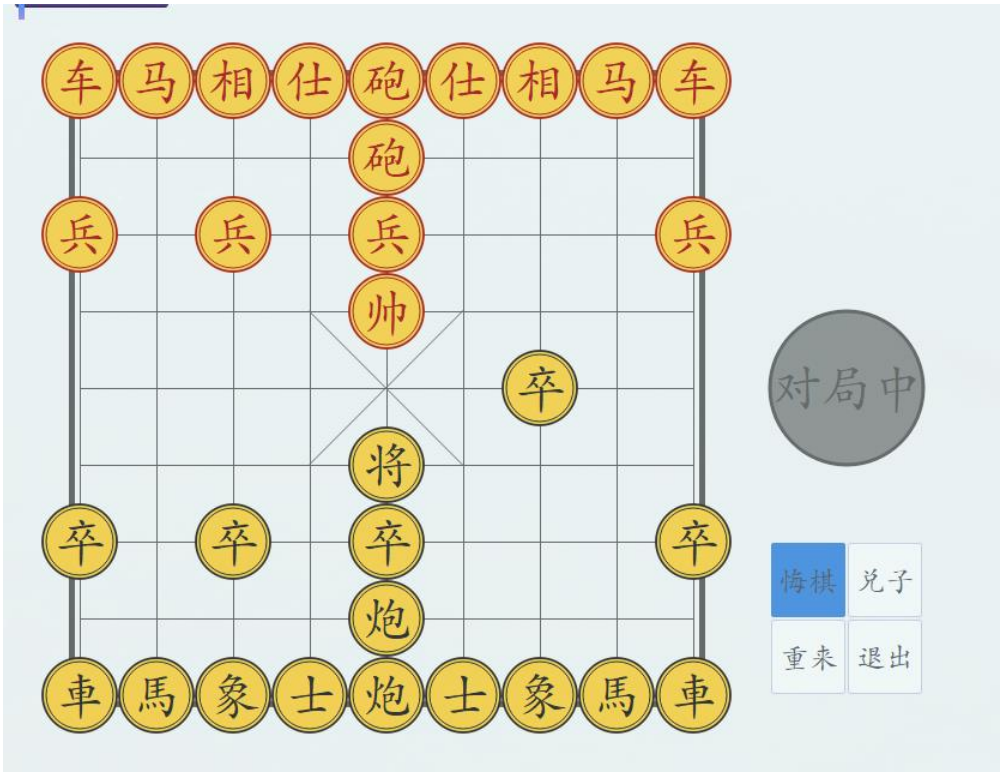

图 3-3-3：悔棋前界面

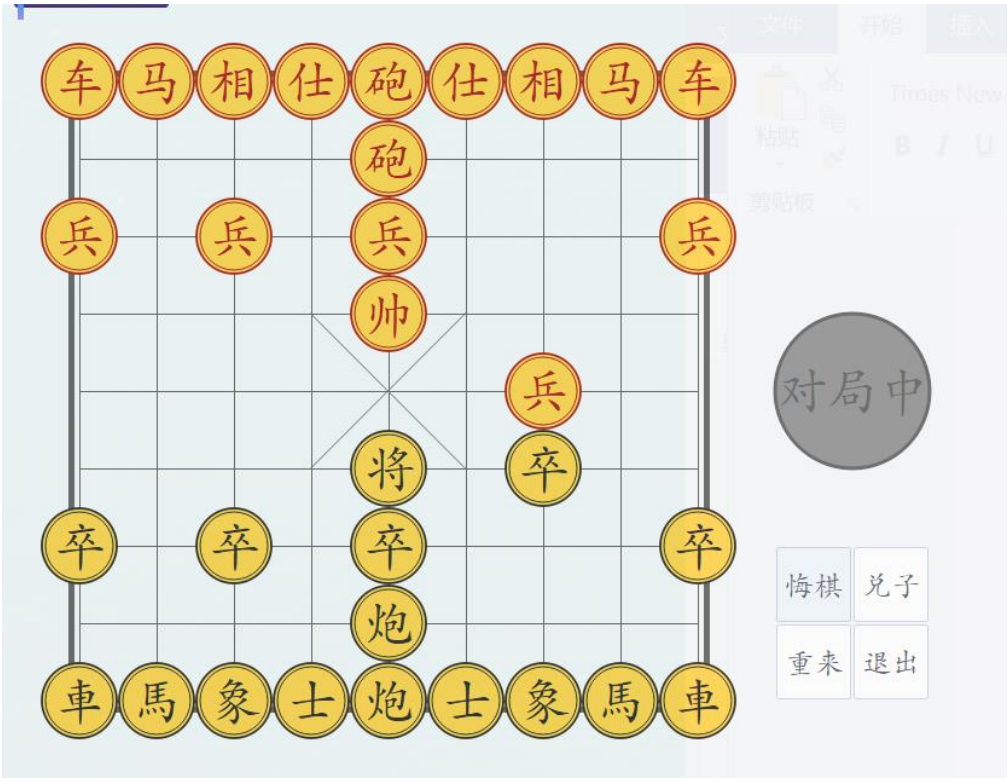

图 3-3-4：悔棋后界面

3.4 棋子兑子功能模块实现

人人对战模块中的兑子功能流程图如图 3-4-1 所示：

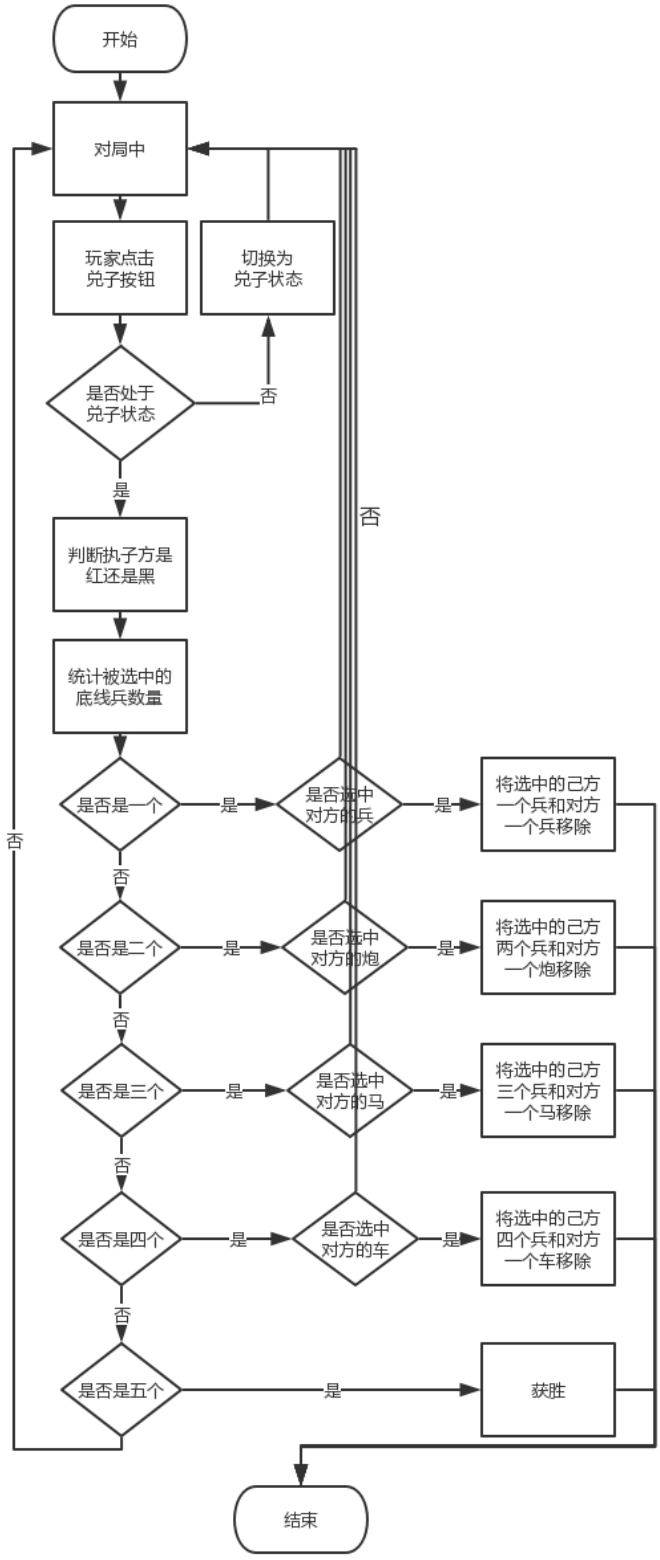

图 3-4-1： 中宫象棋兑子功能流程图

中宫象棋兑子功能核心代码如图 3-4-2 及 3-4-3 所示：

```
void Board::dz()
{
    _dz=!_dz;
    int num=0;
    int kill=0;
    TYPE killType = NO;
    if(_dz)
    {
        for(int i=0;i<32;i++)
        {
            stone[i].push(_s[i]);
        }
        return;
    }
    else if(_bRedTurn)
    {
        for(int i=10;i<=14;i++)
        {
            if(_s[i]._select==true&&_s[i]._row==8)
            {
                _s[i]._dead=true;
                //棋子死后行列要置到棋盘外!!!
                _s[i]._row=100;
                _s[i]._col=100;
                num++;
                qDebug()<<num;
            }
        }
        switch(num)
        {
            case 0:
                for(int i=0;i<32;i++)
                {
                    _s[i]._select=false;
                }
                update();
                return;
            case 1:
                killType = BING;
                break;
            case 2:
                killType = PAO;
                break;
            case 3:
                killType = MA;
                break;
            case 4:
                killType = CHE;
                break;
        }
        if (killType ==BING)
        {
            for(int i=26;i<=30;i++)
            {
               if(_s[i]._select==true)
               {
                   _s[i]._dead=true;
                   //棋子死后行列要置到棋盘外!!!
                   _s[i]._row=100;
                   _s[i]._col=100;
                   kill+=1;
                   break;
               }
            }
        }
        else if(killType == PAO)
        {
            for(int i=24;i<=25;i++)
            {
               if(_s[i]._select==true) {...}
            }
        }
        else if(killType== MA)
        {
            for(int i=18;i<=19;i++)
            {
               if(_s[i]._select==true) {...}
            }
        }
        else if(killType == CHE)
        {
            for(int i=16;i<=17;i++)
            {
               if(_s[i]._select==true) {...}
            }
        }
        for(int i=0;i<32;i++)
        {
            _s[i]._select=false;
        }
        if(kill==0)
        {
            if(stone->length()>0)
                {
                    _selectid=-1;
                    for(int i=0;i<32;i++)
                    {
                        _s[i]=stone[i].top();
                        _s[i]._select=false;
                        stone[i].pop();
                    }
                }
            //没有兑子时不需要改变执子方
            //_bRedTurn=!_bRedTurn;
            update();
            return;
        }
            _selectid=-1;
            _bRedTurn=!_bRedTurn;
            update();
            return;
    }
    //黑方兑子时类似上面红方代码
    else if(!_bRedTurn) {...}
```

图 3-4-2 及 3-4-3：兑子功能核心代码

兑子功能模块中的主要界面如图 3-4-4 及 3-4-5 所示：

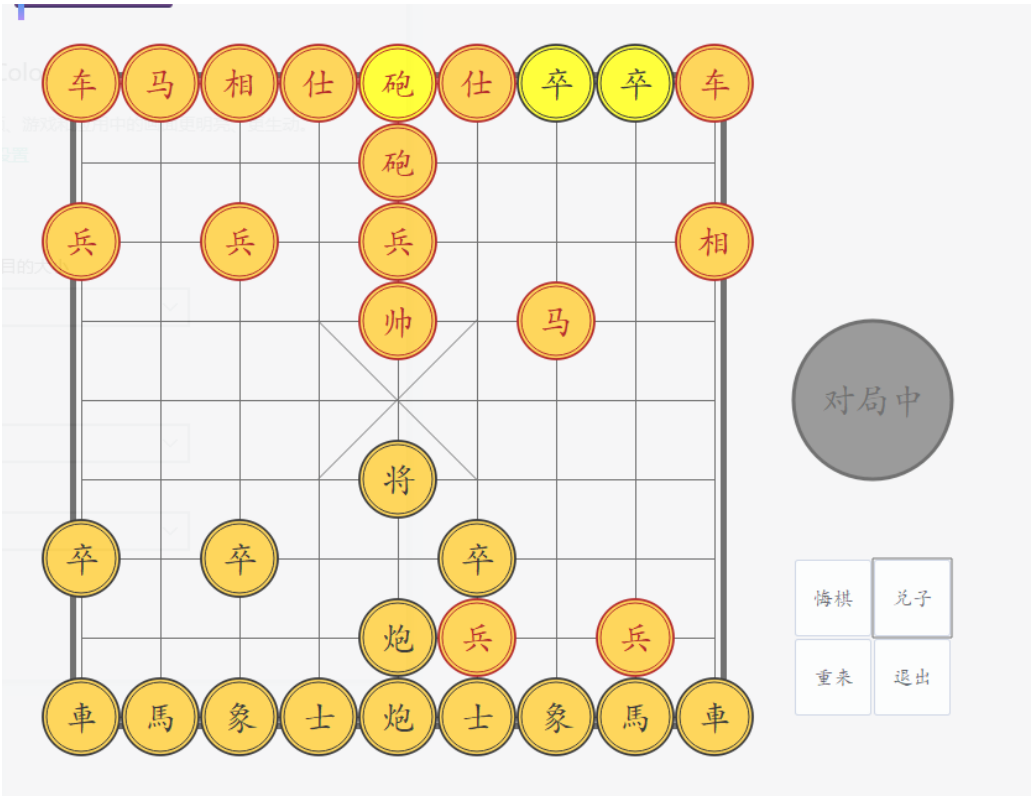

图 3-4-4：兑子前界面

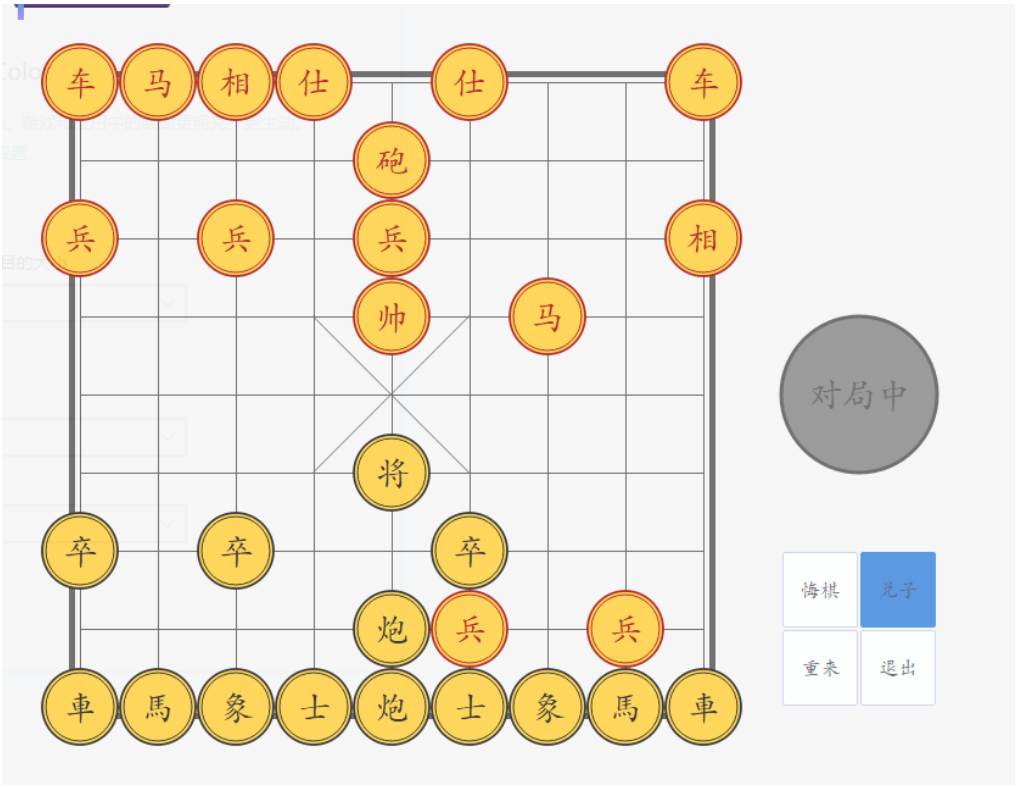

图 3-4-5：兑子后界面

3.5 自动走棋功能模块实现

人机对战模块中的自动走棋功能流程图如图 3-5-1 所示：

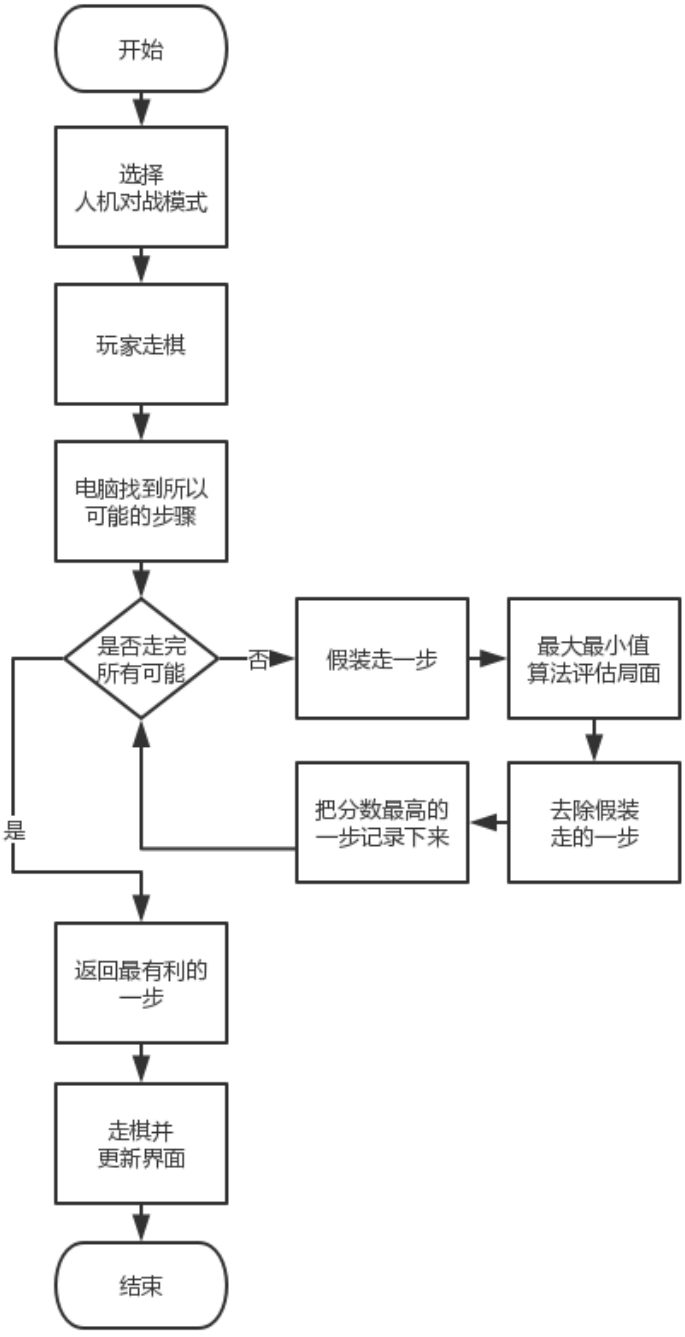

图 3-5-1： 中宫象棋自动走棋功能流程图

中宫象棋自动走棋功能核心代码如图 3-5-2 所示：

```
void SingleGame::computerMove()
{
    //找最有利的走法
    Step* step = getBestMove();
    //走
    moveto(step->_moveid,step->_rowTo,step->_colTo,step->_killid);
    //销毁内存
    delete step;
    //更新界面
    update();
}
Step* SingleGame::getBestMove()
{

    //找到所有可能步骤
    QVector<Step*> steps;
    getAllPossibleMove(steps);
    //尝试走并评估局面
    int maxScore = -100000;
    Step* ret=new Step;
    while(steps.count())
    {
        Step* step = steps.back();
        steps.removeLast();
        //假装走一步
        fakeMove(step);
        //简单算法评估局面
        //int score = calcScore();
        //最大值最小值算法评估局面，其中_level默认为5
        int score = getMinScore(_level-1,maxScore);
        //去除假装走的一步
        unfakeMove(step);
        //找最有利的一步
        if(score > maxScore)
        {
            maxScore = score;
            if(ret) delete ret;
            ret = step;
        }
        else
        {
            delete step;
        }
    }
    //返回最好的结果
    return ret;
}
```

图 3-5-2：自动走棋功能核心代码

程序启动后选择人机对战模式，选择模式界面如图 3-5-3 所示：

图 3-5-3：人机对战模式选择界面

自动走棋功能模块中的主要界面如图 3-5-4 及 3-5-5 所示，人选择兵移动后电脑自动走棋：

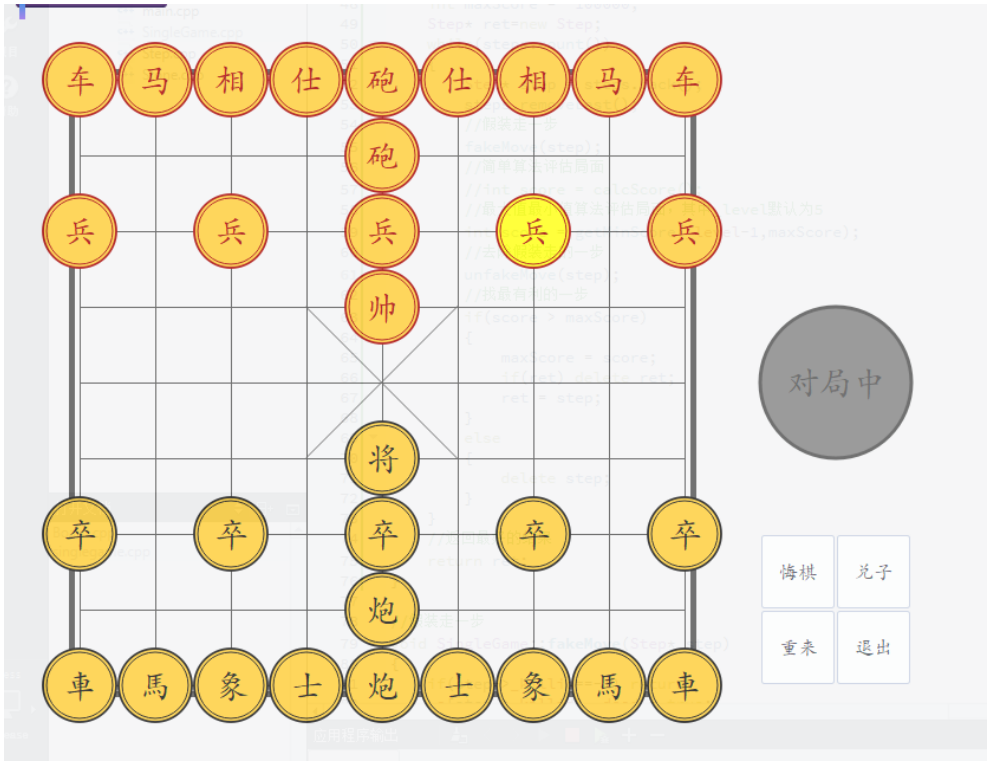

图 3-5-4：自动走棋前界面

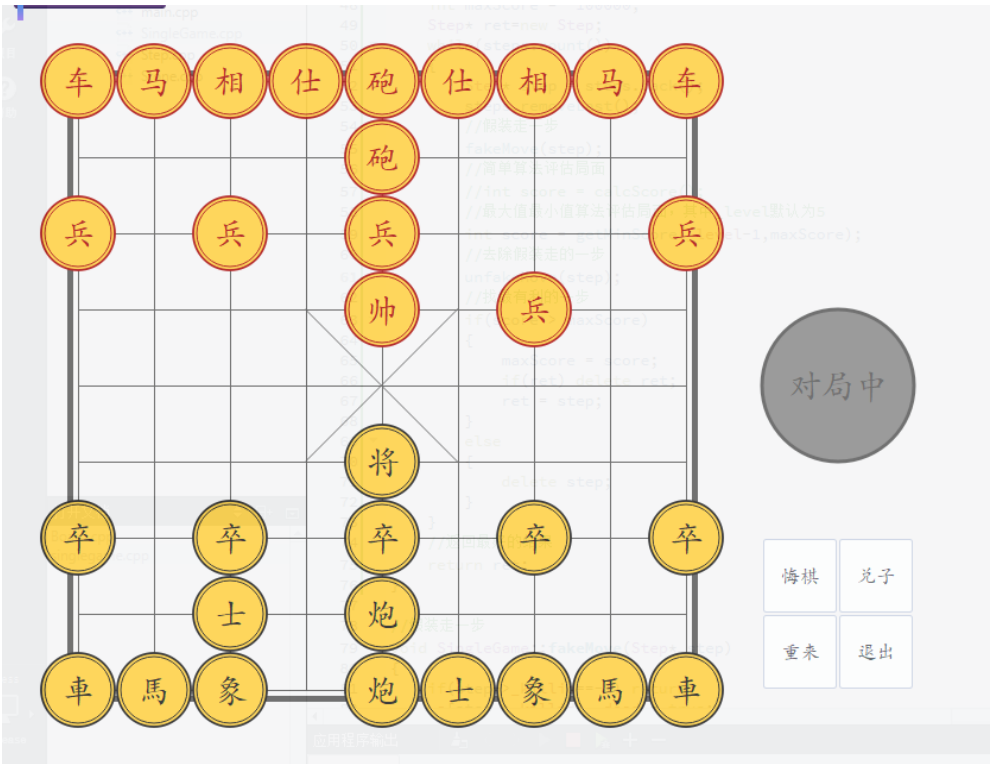

图 3-5-5：自动走棋后界面

3.6 走棋功能模块测试

走棋功能输入测试用例表如表 3-4-1 所示，选不同的棋子重复十次，测试用例全部通过：

表 3-4-1　走棋功能输入测试用例表

序号	测试功能	测试步骤	期望结果	实际结果	是否通过
1	移动棋子	移动自己的棋子,不在自己的回合	棋子不移动	棋子未移动	是
2	移动棋子	移动自己的棋子,在自己的回合,棋子移动不合法	棋子不移动	棋子不移动	是
3	移动棋子	移动自己的棋子,在自己的回合,棋子移动合法	棋子移动	棋子移动	是

3.7 悔棋功能模块测试

悔棋功能输入测试用例表如表 3-5-1 所示，选不同的棋子重复十次，测试用例全部通过：

表 3-5-1　悔棋功能输入测试用例表

序号	测试功能	测试步骤	期望结果	实际结果	是否通过
1	悔棋	不在自己的回合悔棋	对方棋子撤回	对方棋子撤回	是
2	悔棋	在自己的回合悔棋	我方棋子撤回	我方棋子撤回	是
3	悔棋	悔棋 n 次（n>1）	棋子交替撤回 n 次或回到初始棋盘	棋子交替撤回 n 次或回到初始棋盘	是

3.8 兑子功能模块测试

兑子功能输入测试用例表如表 3-5-1 所示，选不同数量的红方的兵和对应的黑方的卒、车、炮、马重复三种用例十次，再用黑方的卒代替兵重复三种用例十次，测试用例全部通过：

表 3-5-1　悔棋功能输入测试用例表

序号	测试功能	测试步骤	期望结果	实际结果	是否通过
1	兑子	选择一个底线兵和一个普通卒进行兑子	该兵和该卒都被移除	该兵和该卒都被移除	是
2	兑子	选择一个底线兵和两个普通卒进行兑子	只有该兵和一卒被移除	只有该兵和一卒被移除	是
3	悔棋	选择两个底线兵和一个普通卒进行兑子	无棋子移除，执子方不变	无棋子移除，执子方不变	是

总　结

本次项目主要实现了中宫象棋的基本游戏规则，通过数组存储棋子、用栈的数据结构存储走过的棋盘状态、容器存储每一次所有可能的棋步，实现了中宫象棋的人人对战、悔棋重来及人机对战等功能。

遇到的问题主要集中在兑子功能代码的编写及电脑自动下棋功能上。

在兑子功能中，判断非常多，导致一个小地方的逻辑不完整可能整个功能都没法用，在重新整理了流程图之后终于以比较方便的方式实现了。

主要出错的地方是在兑子失败后转换了执子方导致悔棋后会出现执子方不对的情况，通过取消执子转换解决。在兑子的过程中也不能够悔棋，否则会导致棋子状态不对，通过在悔棋功能中加入是否正在兑子的判断解决。

在电脑下棋功能中，棋子总是在通个地方来回摇摆，这是因为在循环中相同分数的步骤总是前面的执行，后面的不执行，通过加入一个 0-9 的随机数序列生成使得循环时尽量随机，让相同分数的步骤有相同的可能性被执行，更接近真实的下棋情况。

有待完善的地方是人机对战时的电脑算法还不支持兑子功能，使得对电脑来说兵的价值没有那么高，而整体算法为了追求效率只算到 5 层，相对也没有那么智能，以后可以加入更多智能算法进行完善。

另外图形化界面基本上是用代码直接写的，没有经过画图软件设计，可能比较简陋，如果作为正式的游戏也理应做一些视觉美感上的设计。

参考文献

[1]　王小春. PC 游戏编程（人机博弈）. 重庆：重庆大学出版社，2002.