程序设计训练 第2周大作业报告 ——网络对战国际象棋软件

计86 2017011998 冯卓尔

设计背景

国际象棋是世界上一个古老的棋种。据现有史料记载，国际象棋的发展历史已将近2000年。关于它的起源，有多种不同的说法，诸如起源于古印度、中国、阿拉伯国家等。

本程序实现了国际象棋的基本规则，并且基于QtNetwork实现网络对战功能，使得两个用户能够通过本软件实现国际象棋对战。

程序运行

本项目所使用的是QT GUI框架，它能够提供便捷的UI绘制，同时利用信号-槽机制实现高效的交互。运行时，QT的版本应当高于5.7。使用前，先删去外部的build-chess-Desktop_Qt_5_13_0_clang_64bit-Debug文件夹，并且删去当前目录下的chess.pro.user，防止不同版本编译引发的错误。使用QT Creator打开chess.pro，选择clang作为kit进行编译，能够运行本项目。注意，由于开发操作系统为macOS，因而内部代码中会使用系统适配的API。

注意，由于本项目基于QtNetwork的QTcpSocket，因而对战的双方在网络连接时有着不同的操作要求。先启动程序的一方应该点击QMainWindow中的创建主机选项。点击这个命令以后程序默认通过获取本地的ip地址，事实上就是127.0.0.1，同时，默认的端口port被设置成为8888. 另一方玩家应该通过点击QMainWindow中的连接主机选项，输入创建主机玩家提供的IP和端口，进行连接。连接成功后可以进行对战。

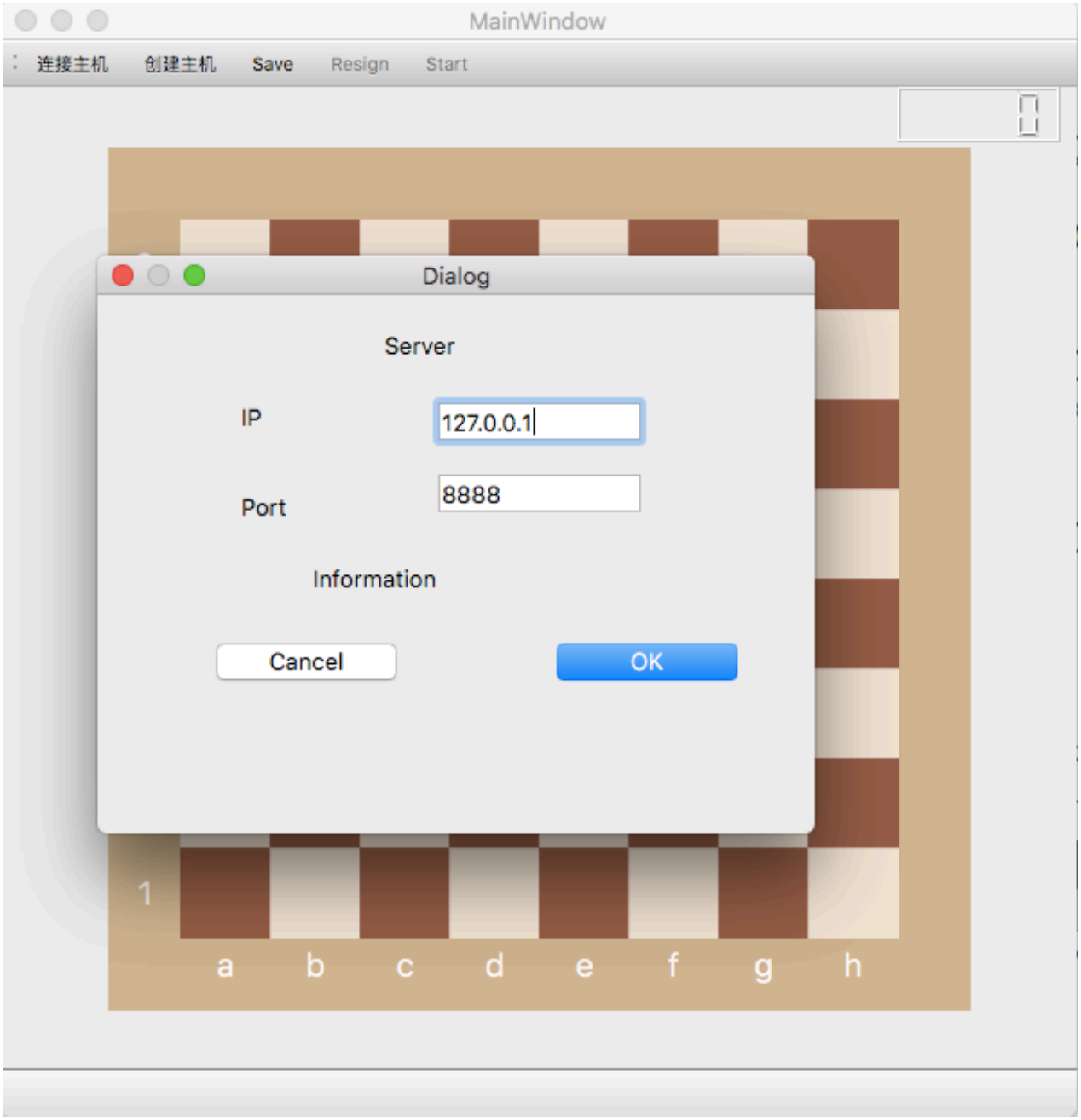
MainWindow
连接主机
创建主机
Save
Resign
Start
Dialog
Server
IP
127.0.0.1
Port
8888
Information
Cancel
OK
1
a
b
c
d
e
f
g
h

图1 主机登录界面

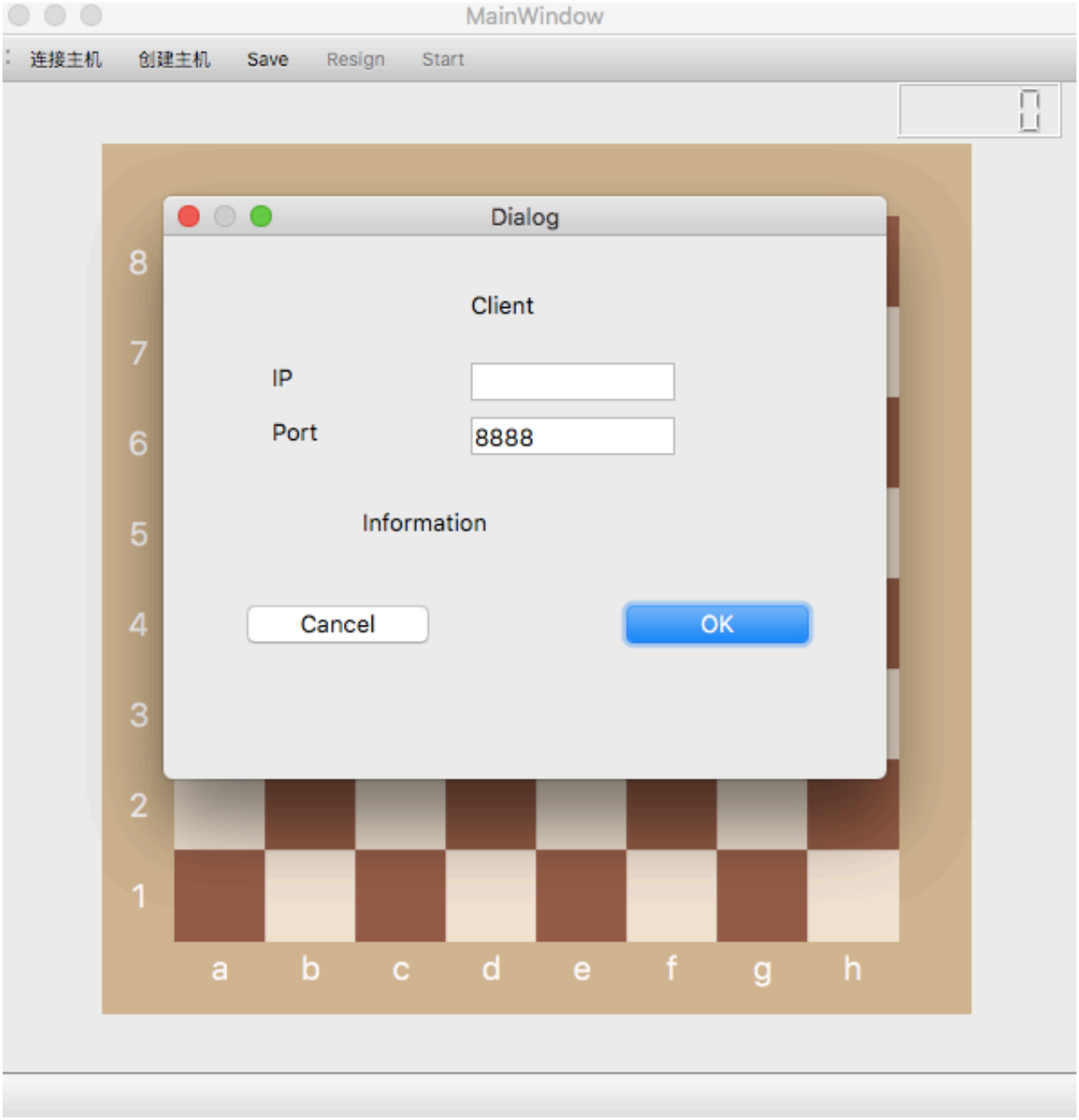

图2 服务端界面

框架设计

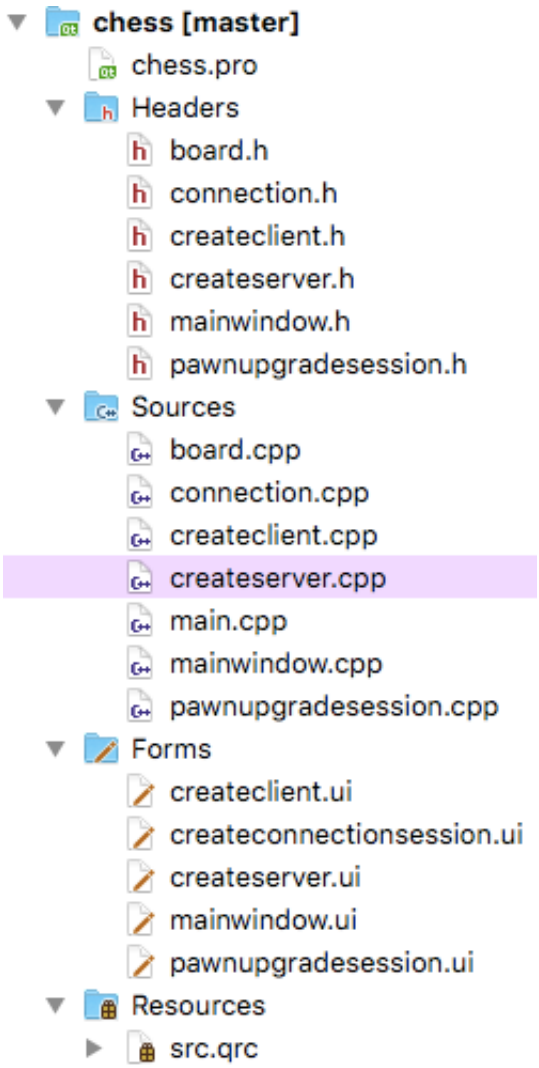

图3 项目框架

本项目结构清晰明了，主要通过两个类Piece和Board来实现逻辑，具体关系如下图。

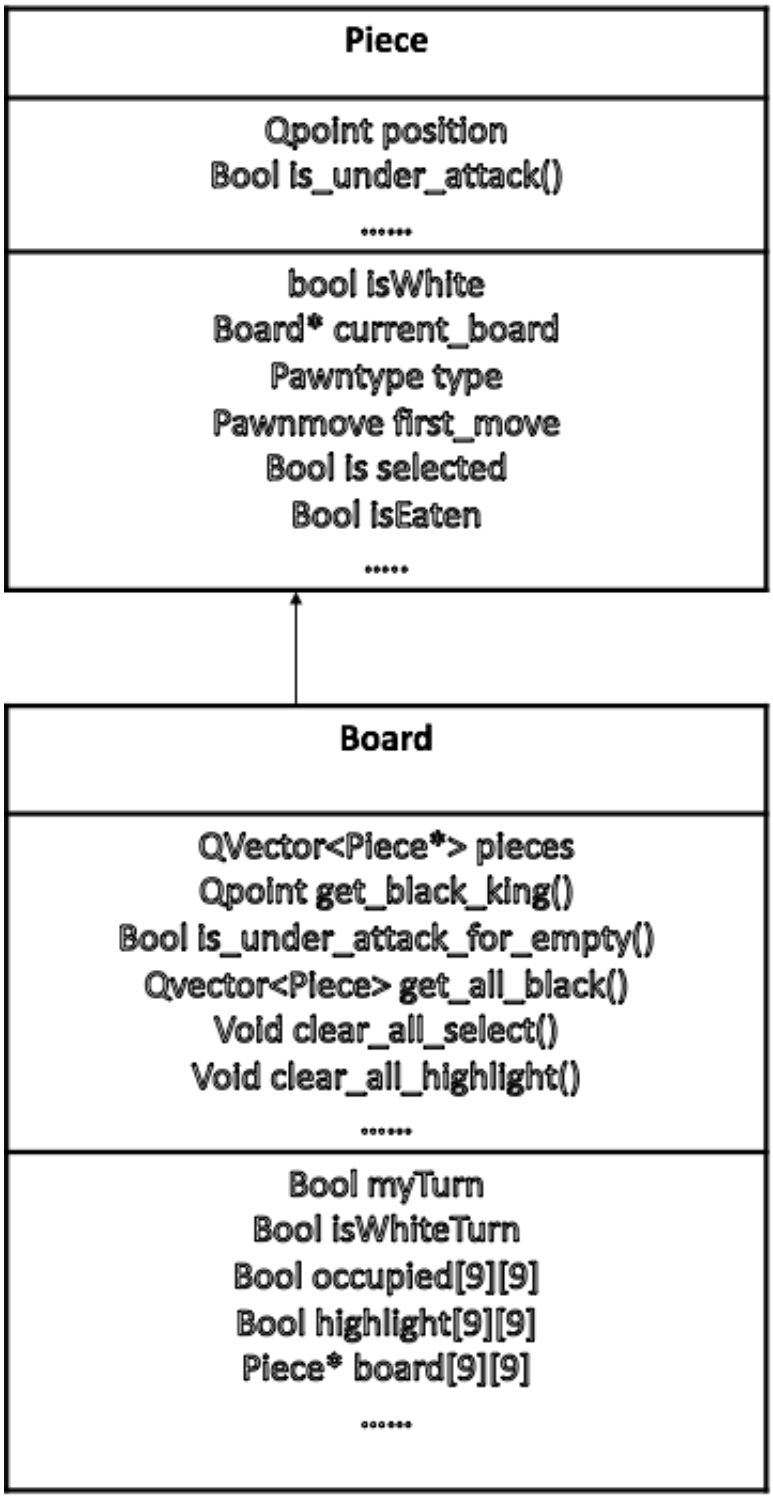

图4 类间关系及主要成员UML

功能说明

根据需求文档，开发者实现了芯片模拟界面以下功能

连接

在实现这个功能的时候，我构建了两个类，一个是`createclient.h` `createserver.h`。
createserver类中比createclient类多一个QTcpServer，两者均通过QTcpSocket进行信息传输。

协议

在棋盘的传输过程中，我们需要一套标准使得服务器端和用户端能够识别互相发送的消息。为了实现有效交互，我设计了一下的传输方案：

```
>>> turn
>>> sync
>>> 棋盘信息
>>> sync_done
>>> timeout
>>> resign
```

其中，turn字段在接收到的时候能够修改当前黑/白方落子，sync是更新棋盘状态的信号，sync_done为停止更新棋盘状态的信号，timeout为对方超时的信号，resign为对方投降的信号。这样子完成了数据的传输。

具体棋盘实现

工程文件中，我新建了`board.h`头文件，里面包含两个类，一个是Piece棋子类，一个是Board棋盘类。其中棋子Piece类主要负责记录具体棋子的位置、类型、初始移动状态等等，而棋盘Board类主要记录选择的棋子位置，双方王的位置以及提供选中棋子后棋子可以到达地点的高亮信息。

为了实现送吃判断，我在具体的棋子中加入了判断当前位置是否收到敌方棋子攻击的函数is_under_direct_attack。由于后续功能需要实现王车易位，我又在Board类中加入了特化的is_under_direct_attack_for_empty函数，用以判断空位置是否受到敌方攻击。

具体的实现效果如图

图5 具体的棋盘高亮以及棋子显示

下面是两个类的具体成员

```
enum PawnMove {NO_MOVE, ONE_MOVE, TWO_MOVE};
enum PieceType {NO_TYPE, PAWN, ROOK, KNIGHT, BISHOP, QUEEN, KING};
enum WinType {WHITE_WIN, BLACK_WIN, FAIR};

class Board;

class Piece {
public:
    Piece();
    Piece(QString& name, QString& place);
    ~Piece();
    PieceType type;
    PawnMove first_move = NO_MOVE;
```

```
    QPoint position;

    bool isWhite = true;
    bool isEaten = false;
    bool isSelected = false;

    bool is_under_direct_attack();

    QVector<QPoint> get_access();

    void do_command(QPoint target);

    Board* current_board = nullptr;
    Board* get_board();

    void move_new(QPoint target);
};

class Board {
public:
    Board();
    Piece* board[9][9];
    bool occupied[9][9];
    bool highlight[9][9];
    void set_highlight(QVector<QPoint>);
    void clear_highlight();
    void clear_all_piece();

    void save_current_board(QTextStream& , bool);

    bool is_under_direct_attack_for_empty(QPoint curp, bool isWhite);

    QVector<QPoint> get_all_black();
    QVector<QPoint> get_all_white();
    QPoint get_black_king();
    QPoint get_white_king();

    bool check_status();

    bool blackLongShiftAble = true;
    bool blackShortShiftAble = true;
    bool whiteLongShiftAble = true;
    bool whiteShortShiftAble = true;

    bool load_board_from_string(QString& info);
    bool init();
};
```

兵的升变

由于Piece类的设计特性，实现兵升变的时候，只需要修改piece的type，改成对应的棋子属性即可。为了方便用户交互，我设计了一个对话框供用户选择升变的类型。具体实现上，我新构建了一个dialog类头文件`PAWNUPGRADESESSION.h`，如果某一个类型为pawn的棋子到达对方底线时，board类判断出其位置弹出对话框，实现升变。

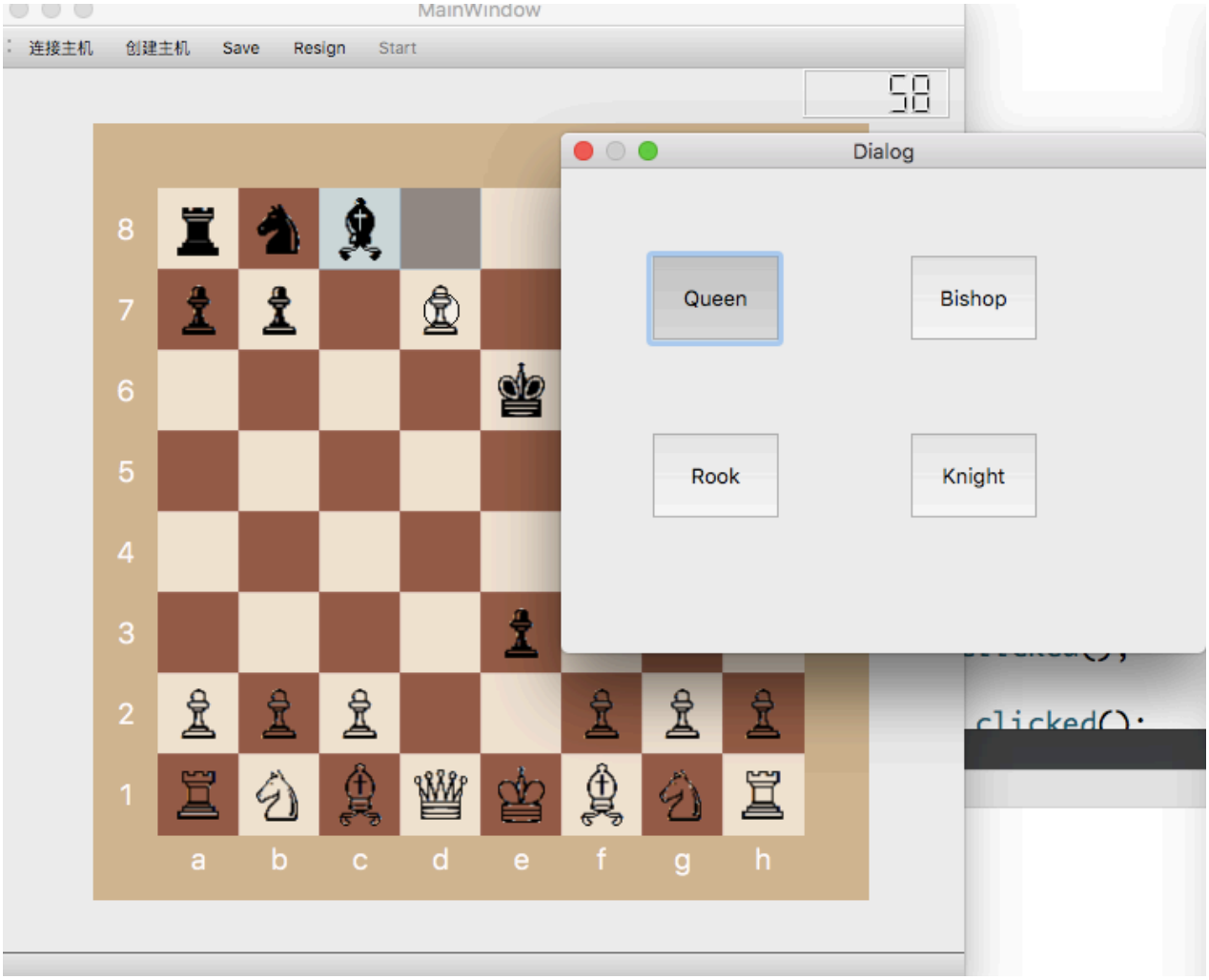

图6 兵的升变

残局绘制和保存

由于我们实现了从文件中读入功能。这个功能能够通过从文件读取文件并且就地生成棋子，将其加入board的当前棋子的QVector中，该部分的具体实现在`board.cpp` read_from_file()中。

保存残局文件的时候，设计师通过同样的方法，根据读入文件的格式，遍历双方棋子并记录，随后将记录结果按照格式写入指定文件。这个方式也是网络请求与发送时候用到的。

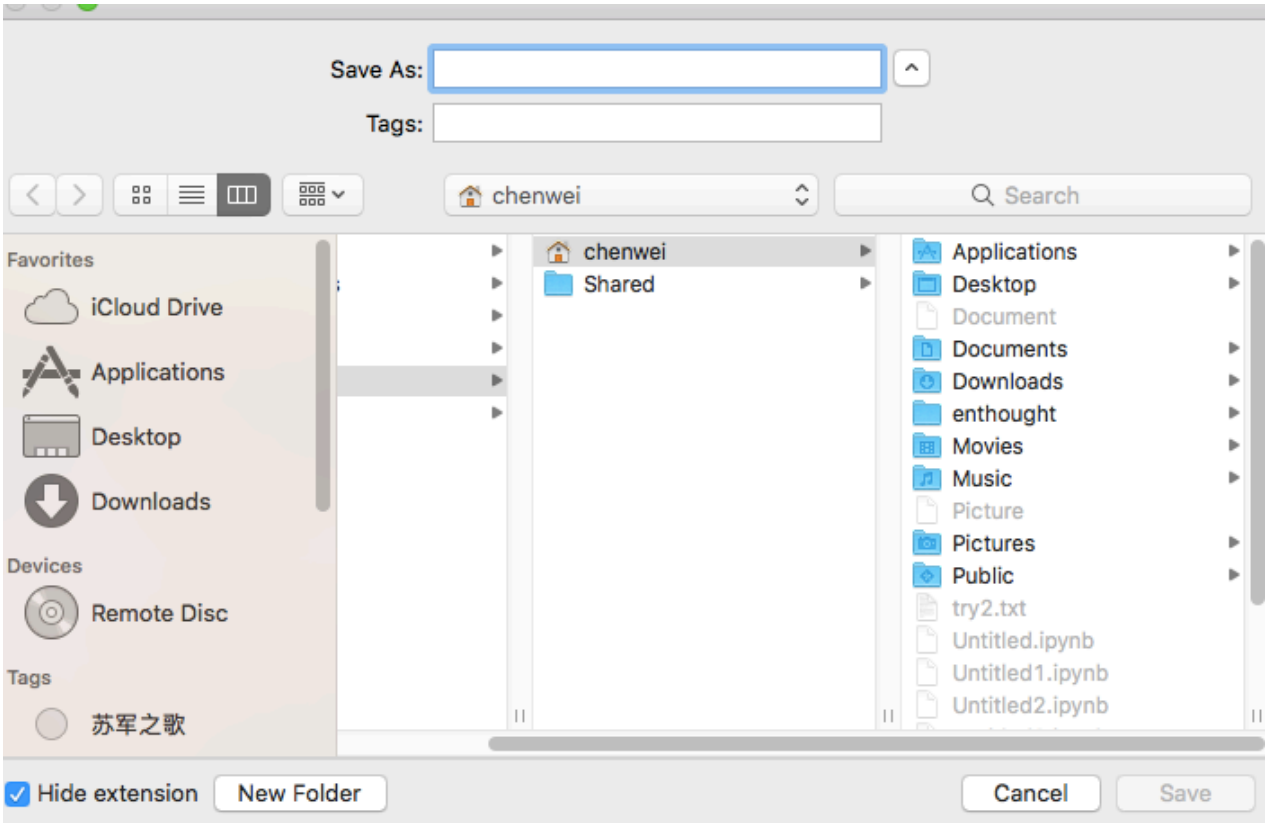

图7 保存文件

王车易位

本程序在实现王车易位的策略是，提供王 王车易位的动作。为了判断王是否能够易位，设计师使用了piece和board中检查路径上所有格子安全度的函数is_under_attack()。随后，复合王和车的移动，便可以实现该功能，具体实现在`borad.cpp`文件move_new()函数中。

王车易位的特殊性带来一个问题，同样的问题是用于兵的第一部移动——如何判断一个棋子是否已经移动过?

设计师使用了修改协议的方法，即对每一个棋子，如果它不在自己的位置上，或者从自己的位置上移动后，网络传输的过程中，会在该棋子的位置编号之后添加一个感叹号，表示已经收到移动。在读入文件/网络信号时，进行两部判断：一部判断棋子是否在原位置上，另一部判断在是否读入了叹号。在棋子中有成员first_move，该成员便是记录这个状态的工具。

图8 王车易位

终局判断

由于我在piece和board中都内嵌了判断当前位置是否安全的函数，因而我在mainwindow类中实现了一个check函数。这个函数能够遍历所有的本方棋子，判断王是否处于攻击状态，也可以判断是否被逼和，具体的实现在`board.cpp` `bool chec_status()` 中。

如果某一方王受到了攻击，系统会弹出对话框进行提醒；如果某一方处于被逼和状态，那么系统会弹出对话框提示双方和棋。


MainWindow
连接主机
创建主机
Save
Resign
Start
60
Black Checked!
OK
8
7
6
5
4
3
2
1
a
b
c
d
e
f
g
h

图9 check！

图10 timeout!

版权说明

本文档说明的工程已经被作者上传至GitHub仓库，如需阅读可以通过用户名的public仓库进行访问。

冯卓尔的GitHub用户名 julius-abu

仓库名称 julius-abu/**chess**

联系方式： RealAndrewFeng@163.com