国际象棋程序设计文档

目录

功能描述

本程序为使用 Qt 5 开发的国际象棋游戏，可在本地进行双人对战。程序实现了较为完整的国际象棋功能。走子上，除基本走子方式外，还支持吃过路兵、王车易位、兵升变，并禁止送将的走子。游戏结束上，实现了将死（checkmate）和无子可动（stalemate）的判定。

程序具有基本的图形界面，可以鼠标单击选中棋子后显示其可走位置，可以通过鼠标单击或拖动进行走子。可以发起求和、认输，或开始新游戏。程序实现了时间条功能，每回合走子存在时间限制，通过时间条直观展示剩余时间，并在时间结束时给出提示。程序还实现了走子记录显示功能，在对局中和游戏结束后查看棋盘每步历史情况，并支持在回放中选中

棋子查看其当时可走位置。

开发构建工具说明

程序使用 Qt Creator 7.0.0(Community)开发，选择 CMake 作为构建系统。源代码包含了 c++的头文件、源文件以及 Qt Creator 项目有关的其它工程文件。程序在 Windows 平台上使用 Desktop Qt 5.15.2 MinGW 64-bit 套件构建了可执行文件。程序开发中仅使用了 c++标准库和 Qt 5 库，原则上源代码是跨平台的，并可在其它平台上构建。如果需要重新构建，可在 Qt Creator 中打开 CMakeLists.txt，Qt Creator 会将相关文件添加进入工程。之后选择工具包即可开始构建。

源代码内容概览

main.cpp 定义了程序的 main()函数，和许多 Qt 程序一样，它实例化一个 QApplication 对象，创建主窗口后启动事件循环。

mainwindow.h,mainwindow.cpp,mainwindow.ui 定义了程序主窗口类 MainWindow，它继承自 QMainWindow 类。.ui 文件是 Qt 的界面文件，用于定义界面的一些外观。

chessview.h,chessview.cpp 定义了棋盘显示小部件类 ChessView，它继承自 QWidget 类，用于在前端图形化界面中展示棋盘情况，接收鼠标点击事件。

chessboard.h,chessboard.cpp 定义了国际象棋的后端组件类 ChessBoard，它继承自 QObject 类，从前端 ChessView 对象接收走子命令，维护棋盘信息，为前端提供绘图数据；以及棋盘数据类 BoardData，该类用 char 数组储存棋盘情况，用来辅助 ChessBoard。

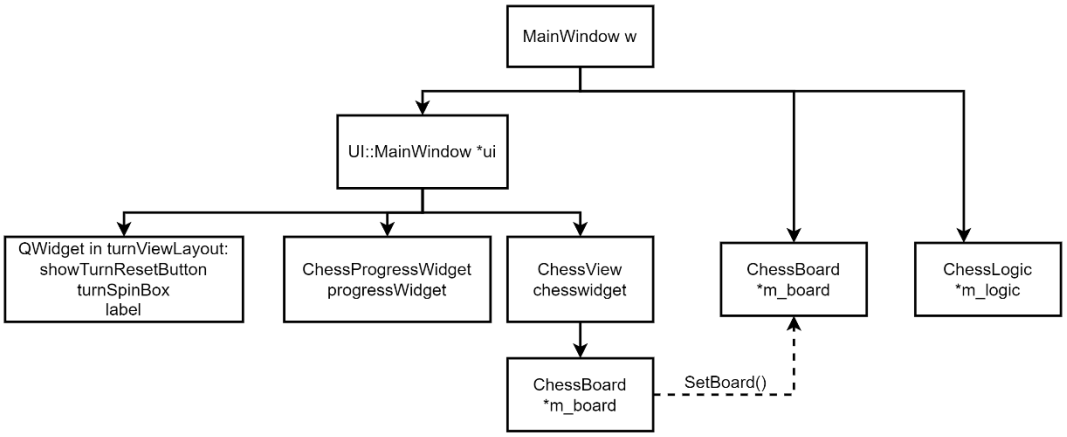
chesslogic.h,chesslogic.cpp 定义了游戏逻辑类 ChessLogic,它继承自 QObject 类。ChessLogic 不是定义走子逻辑的类（走子逻辑在 ChessBoard 中实现），它的功能是处理和转发各组件间的信号。

chessprogresswidget.h,chessprogresswidget.cpp,chessprogresswidget.ui 定义了进度条小部件类 ChessProgressWidget，它继承自 QWidget 类。它用于展示时间条，展示求和、认输、重置游戏的按钮并接受按钮事件。

promotiondialog.h,promotiondialog.cpp,promotiondialog.ui 定义了兵升变的对话框类 PromotionDialog,它继承自 QDialog 类。

img.qrc 是 Qt 资源文件，保存了程序中图标资源的位置。

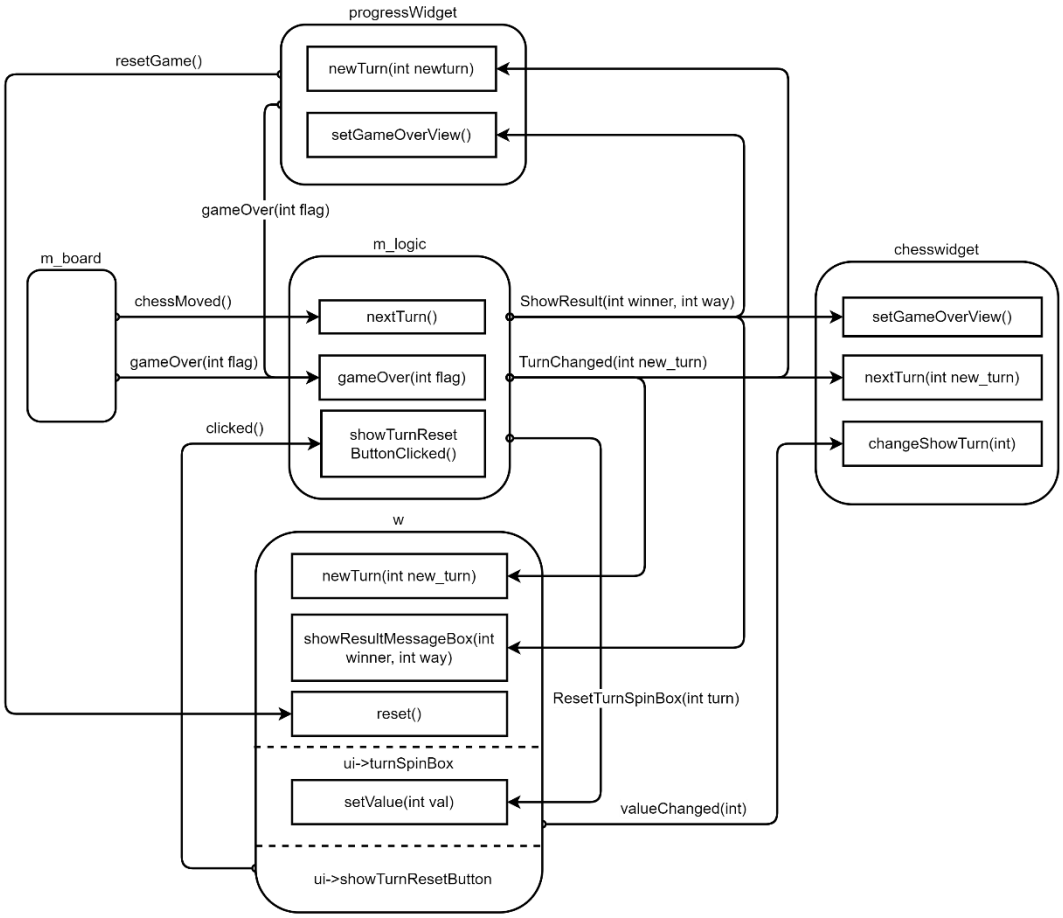
对象结构



程序实例化一个 MainWindow 对象 w，使用 new 创建 UI::MainWindow，ChessBoard 和 ChessLogic 对象并通过指针连接到 w（将 w 作为它们的 Qt 父对象）。ui 包含了前端界面的对象，包括 ChessView 对象 chesswidget，ChessProgressWidget 对象 progressWidget 和一些显示当前步数与历史回放的小部件。

各对象间需要进行通信。由于象棋前端部件 chesswidget 需要和后端部件频繁发送指令和获取棋盘数据，chesswidget 中定义了指向 w->m_board 的指针，用于直接操作 w->m_board。其余对象间的通信使用 Qt 的信号与槽（Signals and Slots）机制实现。

信号流向图



图中展示了主要对象间信号与槽连接的情况，它们在 mainwindow.cpp 中定义。Qt 中信号与槽机制可参阅文档[1]。Qt 约定使用 emit 代表发出信号，在代码中看到 emit 时，可查阅此图快速了解信号流向。有些对象内部还有信号与槽的连接（如 progressWidget 中按键的信号与槽的连接），这些连接没有在图上画出。

代码中一些变量含义的约定

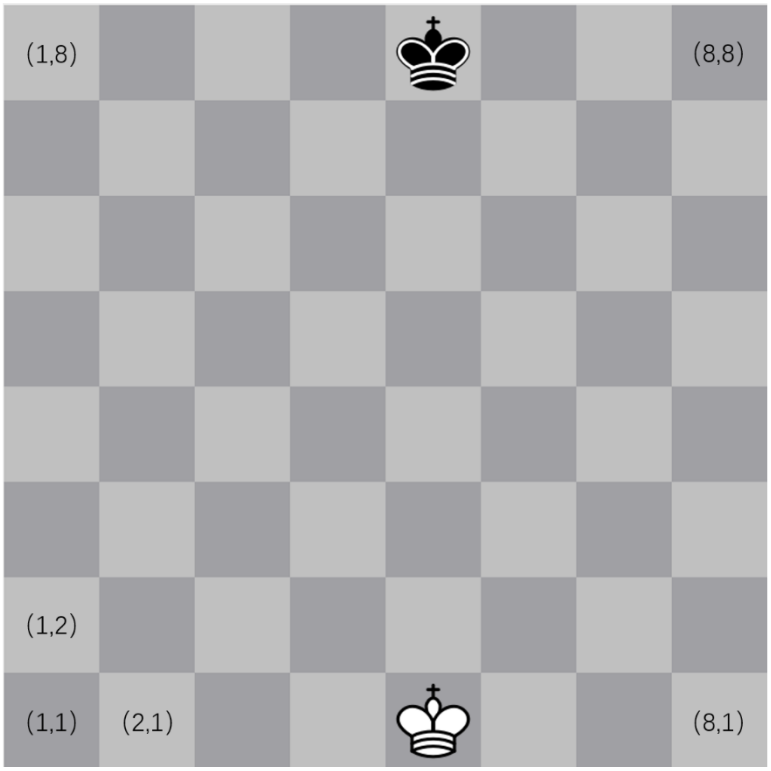
1．轮数（turn）和玩家（player）

代码中以 turn 结尾的整型变量一般表示轮数或步数。游戏开始，双方都未走子时，turn=0，或称为第 0 轮；白方先走，走子后 turn=1，黑方再走子后 turn=2，以此类推。player 用整数 0 或 1 表示，0 代表白方，1 代表黑方。综上，在第 turn 轮，编号为 turn%2 的玩家走子。

[1] https://doc.qt.io/qt-5/signalsandslots.html

2．棋盘坐标

使用二元组 QPoint(col,row)表示棋盘的逻辑坐标。第一个参数 QPoint.x()代表棋盘的列（棋盘上的 a,b,…,h），第二个参数 QPoint.y()代表棋盘的行（棋盘上的 1,2,…,8），两个参数有效取值均为[1,8]间整数，另外 QPoint(0,0)用于代表无效点（此时 QPoint.isNull()返回 true）。



3．棋盘信息

棋盘信息用 BoardData 对象保存，保存在其私有成员变量 char m_data[8][8]中。m_data[c-1][r-1]存储了棋盘坐标 QPoint(c,r)的相关信息，每格的信息为一个字符 char。

程序中用到两套信息。第一套表示棋盘格内的棋子。大写字母代表白方棋子，小写字母代表黑方棋子，棋子种类用其记谱的字母（K,Q,R,B,N,P 和对应小写字母）表示。如'K'代表该格内有白方王，'n'代表该格内有黑方马（骑士）。格中没有棋时用空格' '表示。访问位置无效（访问棋盘外信息）时返回'#'。

第二套表示被选中的棋子可以执行的操作，名字带有 highlight 的 BoardData 对象储存这一套信息（这些信息提供给前端 ChessView 用于绘制高亮格）。同样用大、小写字母代表白、黑方。具体字母含义见下表。

A,a	该位置棋子被选中

M,m	被选中棋子可以移动到此（包括移动后吃掉此处棋子）
C,c	移动到此会导致送将，因此禁止
R,r	兵移动到此升变
E,e	兵移动到此吃对方过路兵
Y,y	王移动到此，完成王车易位

4．游戏结束的标志（flag,winner,way）

这里有两组标志。一组是 m_board 和 progressWidget 在游戏结束或求和认输后发出信号 gameOver(int flag)中的 flag,可以看做是原始标志；另一组是 m_logic 的 gameOver(int)槽发出信号 ShowResult(int winner,int way)中的 winner 和 way，可以看做处理后的标志，用于各组件展示结果。

flag

1	白方将死黑方（白胜）
2	黑方将死白方（黑胜）
3	白方走子但无子可动（平局）
4	黑方走子但无子可动（平局）
5	双方约定后和棋
6	白方投子认负
7	黑方投子认负
0	错误：未找到黑王
-1	错误：未找到白王
-2	错误：两王均未找到
-3	未知错误

winner

0	白胜
1	黑胜
2	平局；白方逼平黑方（白方无子可动）
3	黑方无子可动

way

0	将死

1	无子可动
2	约定后和棋
3	投子认负
-1	错误：未找到王

各类的详细文档

MainWindow Class

Header	#include "mainwindow.h"
Inherits	QMainWindow

MainWindow 是程序的主窗口类，它包含指向 UI::MainWindow、ChessBoard 和 ChessLogic 对象的指针，在构造函数中实例化这些对象，并成为它们的 Qt 父对象。

public:

MainWindow(QWidget *parent = nullptr);

构造函数，用 new 方法实例化 UI::MainWindow、ChessBoard 和 ChessLogic 对象，将指针赋给自己的私有成员变量，连接各组件的信号和槽，以及其它初始化。

~MainWindow();

析构函数

public slots:

void showResultMessageBox(int winner, int way);

接收游戏结果信号后弹出一个 QMessageBox 显示游戏结果。

void newTurn(int new_turn);

接收轮数更新信号后，更新 ui->turnSpinBox 和 ui->turnLabel（即界面顶部显示轮数组件）显示信息。

void reset();

接收重置游戏信号后，删除原来 ui,m_board 和 m_logic 对象，重新实例化它们并初始化。

private:

void connectSignalToSlot();

连接组件的信号和槽。

Ui::MainWindow *ui;

存放在构造函数和 reset()中实例化的 Ui::MainWindow 对象指针。

ChessBoard *m_board;

存放在构造函数和 reset()中实例化的 ChessBoard 对象指针。

ChessLogic *m_logic;

存放在构造函数和 reset()中实例化的 ChessLogic 对象指针。

ChessView Class

Header	#include "chessview.h"
Inherits	QWidget

ChessView 类用于绘制并展示棋盘情况，它还负责接收发生在棋盘区域的鼠标点击和拖动事件。ChessView 类有一个重要的私有成员变量 m_board,它是指向 ChessBoard 对象的指针。ChessView 对象从该指针指向的 ChessBoard 对象获取绘图数据，并将鼠标事件转换为走子坐标后传给该 ChessBoard 对象。

public:

ChessView(QWidget *parent = nullptr);

构造函数，初始化私有成员变量。

void setBoard(ChessBoard *chessboard);

为 ChessView 绑定一个 ChessBoard 对象。让私有成员变量 m_board 指向 *chessboard。

public slots:

void nextTurn(int new_turn);

接收轮数更新信号，刷新绘制。

void changeShowTurn(int show_turn);

接收需要绘制的轮数 show_turn，从 m_board 中拿数据并绘制。根据绘制的轮数是否为最新设置 isShowHistory 标志。

void setGameOverView();

接收游戏结束信号，设置 gameIsOver 标志。

protected:

void drawGrid(QPainter *painter);

绘制棋盘格，参看 paintEvent();

void drawPieces(QPainter *painter, BoardData *data);

绘制棋子,但不绘制正在被拖动的棋子,参看 paintEvent(),drawDrag(),dragOn，m_chosedPiece。

void drawHighLight(QPainter *painter, BoardData *highlightmap);

绘制高亮格。高亮的格子用于提示被选中的棋子可能的行动。参看 paintEvent()。

void drawDrag(QPainter *painter, BoardData *data);

绘制正在被拖动的棋子，参看 paintEvent()，dragOn，m_chosedPiece。

QRect gridRect(int column, int row);

坐标转换。输入行数和列数（即棋盘坐标），返回棋盘格在绘图窗口中的绘图坐标区域（用 QRect 对象，即一个矩形表示）。参看 eventGrid()，棋盘坐标。

QPoint eventGrid(QPoint eventpoint);

坐标转换。输入绘图窗口中的绘图坐标，返回棋盘坐标（即几行几列）。参看 gridRect(),棋盘坐标。

int pieceplayer(char piece);

输入字符 piece（通常是 BoardData 中某个格子的信息），返回其对应 player 编号。白方返回 0,黑方返回 1,都不是(如 piece 为' '或'#')返回-1。参看玩家(player)，棋盘信息，BoardData。

void paintEvent(QPaintEvent *event);

重写 QWidget 的虚函数。绘制 ui。参看 drawGrid(),drawHighLight(),drawPieces(),drawDrag()。

void mousePressEvent(QMouseEvent *event);

重写 QWidget 的虚函数。当有鼠标点击事件时会被调用，处理鼠标按下事件。如果鼠标选中了新棋子，就让 m_board 在 m_board->m_highLightMap 中填入选中棋子可执行操作的信息，以供 paintEvent()绘制使用。如果再次选中该棋子，则启用拖动标志 dragOn。如果选中了有 highlight 的格子，则根据信息处理之前选中棋子的移动等。参看棋盘信息，paintEvent()，m_chosedPiece，dragOn。

void mouseReleaseEvent(QMouseEvent *event);

重写 QWidget 的虚函数。在鼠标释放时会被调用，处理鼠标释放事件。当拖动标志 dragOn 为 true 时，此处可以获取结束拖动的位置，并做相应处理。

void mouseMoveEvent(QMouseEvent *event);

重写 QWidget 的虚函数。在鼠标按下到释放前会被连续调用，处理鼠标按下后的移动事件。当拖动标志 dragOn 为 true 时，此处需要更新储存鼠标位置的私有成员变量 mousePoint。

void resizeEvent(QResizeEvent *event);

重写 QWidget 的虚函数。窗口大小改变时会被调用，更新储存棋盘格大小的私有成员变量 gs。

QSize sizeHint() const;

重写 QWidget 的虚函数。

QSize minimumSizeHint() const;

重写 QWidget 的虚函数。

private:

int gs;

棋盘格大小。

QMap<char, QIcon> m_piece;

棋子的图标,以键值对形式存放。

ChessBoard *m_board;

存放指向 ChessBoard 对象的指针，ChessView 对象从该指针指向的 ChessBoard 对象获取绘图数据，并将鼠标事件转换为走子坐标后传给该 ChessBoard 对象。

bool chessChosed;

有棋子被选中时此标志为 true。参看 mousePressEvent()。

bool dragOn;

拖动标志。参看 mousePressEvent()，mouseReleaseEvent()，mouseMoveEvent()。

QPoint mousePoint;

在拖动启用时,存放鼠标位置,供绘制拖动中的棋子使用。参看 mouseMoveEvent()，drawDrag()。

QPoint m_chosedPiece;

储存被选中棋子的棋盘坐标。参看棋盘坐标，mousePressEvent()。

int m_drawturn;

需要绘制的轮数。参看轮数（turn）。

int m_maxturn;

当前游戏进行的轮数，也是最大的绘制轮数。参看轮数（turn）。

bool gameIsOver;

游戏结束前此标志为 false,结束后被 setGameOverView()置为 true。此标志为 true 时，ChessView 不会启动拖动和调用 m_board->moveChess()。参看 setGameOverView()，dragOn，mousePressEvent()。

bool isShowHistory;

当 m_drawturn 不等于 m_maxturn 时，即需要绘制的不是最新一轮棋盘时，此标志为 true，此时 ChessView 不会启动拖动和调用 m_board->moveChess()。参看 changeShowTurn()，dragOn，mousePressEvent()。

ChessBoard Class

Header	#include "chessboard.h"
Inherits	QObject

ChessBoard 是国际象棋的后端组件类，它从前端 ChessView 对象接收走子命令，维护棋盘信息，为前端提供绘图数据。

该类用一个有序容器储存一系列 BoardData 对象，即私有成员变量 QVector<BoardData> m_boardhistory，这些 BoardData 对象储存了每一轮棋盘上的棋子位置。该类的另两个私有成员变量 BoardData m_chessMap 和 BoardData m_highLightMap，分别储存了当前轮棋盘上的棋子位置和被选中棋子的可执行操作。

ChessBoard 类的工作流程大致为：前端 ChessView 对象调用公共方法 updateHighLightMap()让 ChessBoard 对象更新 m_highLightMap；ChessView 对象调用 moveChess()方法，ChessBoard 对象根据 m_highLightMap 信息更改 m_chessMap 信息（走棋），将更新后的 m_chessMap 添加进 m_boardhistory 的末尾（储存对局历史），然后发出 chessMoved()信号；走棋后检查游戏是否结束，如果结束发出 gameOver(int flag)信号。此外，ChessView 对象可通过公共方法 getBoardData(int turn)获取 m_boardhistory 数据，用 getHighLightMap()方法获取 m_highLightMap 数据，用于绘图。

public:

ChessBoard(QObject *parent = nullptr);

构造函数。创建一个 BoardData 对象，写入初始局面，将其赋给 m_chessMap，并添加到 m_boardhistory（为第 0 步的棋盘情况）。

BoardData *getBoardData(int turn);

返回第 turn 轮的棋盘数据，其记录了棋子的位置。

BoardData *getHighLightMap();

返回 m_highLightMap 的指针。

void resetHighLightMap();

将 m_highLightMap 中信息清空。

void updateHighLightMap(QPoint eg, int turn);

在第 turn 轮的棋盘数据中，将棋盘坐标 eg 的棋子作为选中棋子，将其可以执行的操作记录在 m_highLightMap 中。参看棋盘信息。

bool moveChess(QPoint startat, QPoint moveto);

在 m_chessMap 中，尝试执行 startat 到 moveto 的走棋操作，根据 m_highLight 中的信息处理特殊操作（兵升变、过路兵、王车易位）。走棋成功返回 true，失败返回 false。

bool getGameOverCheckingFlag();

返回 gameOverChecking。

signals:

void chessMoved();

moveChess()成功走棋时发出此信号。

void gameOver(int flag);

checkGameOver()中判定游戏结束时发出此信号，结束标志 flag 参看游戏结束的标志（flag,winner,way）。

protected:

int pieceplayer(char piece);

返回 piece 对应 player 编号。白方返回 0，黑方返回 1，都不是（如 piece 为' '或'#'）返回-1。参看玩家（player），棋盘信息。

void highLight_P(int turn, int col, int row);

对（col,row）位置的白兵，根据白兵走子规则初步更新 m_highLightMap，不检查送将的操作。参看 updateHighLightMap()，verifyHighLightMap()。

```
void highLight_p(int turn, int col, int row);
void highLight_K(int turn, int col, int row);
void highLight_k(int turn, int col, int row);
void highLight_R(int turn, int col, int row);
void highLight_r(int turn, int col, int row);
void highLight_B(int turn, int col, int row);
void highLight_b(int turn, int col, int row);
void highLight_Q(int turn, int col, int row);
void highLight_q(int turn, int col, int row);
void highLight_N(int turn, int col, int row);
void highLight_n(int turn, int col, int row);
```

参看 highLight_P()，updateHighLightMap()。

```
bool gridControlledByPlayer(int player, BoardData chessBoard, int col, int row);
```

chessBoard 为记录了棋子位置的 BoardData 对象，检查（col,row）位置是否在 player 棋子的控制中（即若此时（col,row）有（1-player）方棋子，player 先手，能否吃掉（col,row）处的棋子），若在控制中则返回 true，否则返回 false。

```
void checkGameOver();
```

在棋子移动后，检查游戏是否已经满足结束条件，若是发出相应信号。参看 moveChess()，gameOverChecking，gameOver(int flag)。

```
int checkKingExist();
```

checkGameOver()的辅助方法。检查 m_chessMap 中双王是否存在，都存在返回 1，仅存在白王返回 0，仅存在黑王返回-1，都不存在返回-2。在正常运行时，该方法应返回 1。

```
bool checkNoLegalMove();
```

checkGameOver()的辅助方法。当前走子方无合法移动方式时返回 true，否则返回 false。

```
int checkCheck(int player, BoardData chessMap);
```

checkGameOver()的辅助方法。检查 chessMap 中 player 方的王是否被将。是则返回 1，否则返回 0。若没能找到王返回-1（发生错误）。

void verifyHighLightMap(int player, BoardData chessMap, QPoint eg);

将初步标记的 m_highLightMap 中，会导致送将的操作标记为'C'（白方操作）或'c'（黑方操作）。参看 updateHighLightMap()，highLight_P()，棋盘信息。

private:

QVector<BoardData> m_boardhistory;

容器储存了一系列 BoardData 对象，它们储存了每一轮棋盘上的棋子位置。参看 BoardData,棋盘信息。

BoardData m_highLightMap;

保存被选中棋子的可执行操作信息。它不用于长期保存信息，更类似于草稿，因为用户会不停切换选中的棋子，verifyHighLightMap() 也将临时信息记录在 m_highLightMap。参看 updateHighLightMap(),resetHighLightMap(),棋盘信息。

BoardData m_chessMap;

保存了当前轮棋盘上的棋子位置。走子时，会先在 m_chessMap 上做更改，让后将更新后的 m_chessMap 添加进 m_boardhistory。参看 moveChess()，棋盘信息。

QVector<bool> castleFlag;

王车易位标志，用于王走子中王车易位的判断。容器储存四个 bool 值，分别对应白王长、短易位和黑王长、短易位。false 代表对应王或车已经移动过或车已经被吃掉，从而该局游戏中不能进行对应易位。注意，即使标志为 true，也不代表此时可以易位（可能因为中间有棋子，所经过格子被对方控制等不能易位），只是说明本局中还有进行某个易位的可能。参看 highLight_K()，highLight_k()。

QVector<QVector<bool>> flagHistory;

储存每一轮王车易位标志的情况，其更新方式类似于 m_boardhistory。

bool gameOverChecking;

checkGameOver() 方法开始和返回时会更改此标志的值。值为 true 代表 checkGameOver() 方法还在进行中。此标志通过公共方法 getGameOverCheckingFlag()提供给前端 ChessView 对象,防止 checkGameOver() 返回前传入意外的输入。

BoardData Class

Header	#include "chessboard.h"
Inherits	None

棋盘数据类，该类用私有成员变量 m_data，即一个 char 数组储存棋盘情况，用来辅助 ChessBoard。储存信息含义参看棋盘信息。

public:

BoardData();

构造函数。初始化私有成员变量 m_data，即用’ ’填充。

char getData(int col, int row);

获取 m_data 中储存的，棋盘坐标为(col,row)的格子的信息。参看棋盘坐标。

void setData(char ch, int col, int row);

将 m_data 中，棋盘坐标(col,row)的格子的信息设置为 ch。参看棋盘坐标。

void reset();

清空 m_data 中的信息，用’ ’填充。

private:

char m_data[8][8];

m_data[c-1][r-1]存储了棋盘坐标(col,row)的格子的相关信息,每格的信息为一个字符。程序中使用到两套信息，一套代表棋盘上棋子的位置，另一套代表被选中棋子可以执行的操作。参看棋盘信息。

ChessLogic Class

Header	#include "chesslogic.h"
Inherits	QObject

游戏逻辑类。ChessLogic 不是定义走子逻辑的类（走子逻辑在 ChessBoard 类中实现），它的功能是处理和转发各组件间的信号。参看信号流向图。

public:

ChessLogic(QObject *parent = nullptr);

构造函数。

public slots:

`void nextTurn();`

更新私有成员变量 `m_turn`，发出 `TurnChanged(m_turn)` 信号。

`void gameOver(int flag);`

根据 `flag`，发出对应 `ShowResult(int winner, int way)` 信号。参看游戏结束的标志（`flag,winner,way`）。

`void showTurnResetButtonClicked();`

发出信号 `ResetTurnSpinBox(m_turn)`。

signals:

`void TurnChanged(int new_turn);`

游戏轮数更新信号，向各组件通知新的轮数。

`void ShowResult(int winner, int way);`

游戏结束后，通知各组件显示结果。

`void ResetTurnSpinBox(int turn);`

连接到 `w->ui->turnSpinBox`，通知其显示最后一轮。

private:

`int m_turn;`

储存了游戏轮数。

ChessProgressWidget Class

Header	#include "chessprogresswidget.h"
Inherits	QWidget

进度条小部件。实例化一个 `QTimer` 对象用于回合计时，绘制时间条展示剩余时间，提供暂停按钮和设置每回合时间的按钮（并实现对应功能）。该小部件还提供求和、认负、重置游戏的按钮，接收按钮事件后发出相应信号。下图展示了其 `ui` 界面情况。

public:

`ChessProgressWidget(QWidget *parent = nullptr);`

构造函数。初始化 `ui` 和其它内部变量，连接内部按钮信号到保护槽。

`~ChessProgressWidget();`

析构函数。

public slots:

void newTurn(int newturn);

更新 ui->playerLabel 并重置计时。

void setGameOverView();

停止计时，禁用求和投降等按钮。

signals:

void resetGame();

重置游戏按钮按下并确认后，发出该信号。参看信号流向图。

void gameOver(int flag);

求和或投降被确认后，发出该信号。参看信号流向图。

protected:

void drawPlayerIcon(QPainter *painter);

绘制当前玩家图标，图标使用该方王的图标。参看 paintEvent()。

void drawTimeBar(QPainter *painter);

绘制时间条，显示回合剩余时间占回合总时间比例。

void showRemainTime();

设置 ui->timeLabel 显示剩余时间。

void paintEvent(QPaintEvent *event);

重写 QWidget 的虚函数。绘制 ui。

void resizeEvent(QResizeEvent *event);

重写 QWidget 的虚函数。在窗口大小改变时会被调用，重新设置各 ui 对象绘图位置。

protected slots:

void timeUpdate();

接收计时器 timeout()信号，更新剩余时间 remain_time。

void pauseButtonClicked();

接收暂停按钮 clicked()信号，停止计时器。

void settingButtonClicked();

接收设置按钮 clicked()信号，从对话框中获取新的每回合时间，并更新 total_time 和 remain_time。

void resetButtonClicked();

接收游戏重置按钮 clicked()信号，弹出对话框进行确认。若确定重置游戏则发出 resetGame()信号。

void offerDrawButtonClicked();

接收求和按钮 clicked()信号，弹出对话框进行确认。若确认和棋按结果发出 gameOver()信号。

void admitDefeatButtonClicked();

接收认负按钮 clicked()信号，弹出对话框确认。若确认投子认负，发出对应 gameOver()信号。

private:

Ui::ChessProgressWidget *ui;

存放构造中实例化的 ui 对象指针。

QMap<int, QIcon> m_playerIcon;

键值对形式存放的双方玩家图标，默认为双方王的图标。key 为玩家编号，value 为对应图标。参看玩家（player）

bool isPause;

暂停标志，如果时间条暂停，此标志为 true。

bool isTimeOut;

超时标志，当前玩家超时后此标志为 true。

int timeInterval;

计时器发出 timeout()信号的周期（ms），也是组件中时间更新的周期。默认为 50。

long remain_time;

回合剩余时间（ms）。

long total_time;

回合总时间（ms）。

QTimer *m_timer;

存放构造函数中实例化的计时器 QTimer 对象的指针。

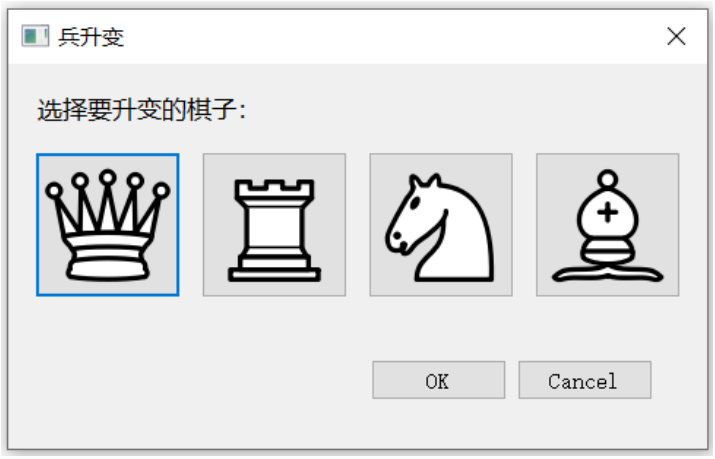
int player;

当前玩家编号。

PromotionDialog Class

Header	#include "promotiondialog.h"
Inherits	QDialog

兵升变的对话框，用于选择兵升变到哪个棋子。ui 如图所示。



public:

PromotionDialog(int player = 0, QWidget *parent = nullptr);

构造函数，player 为将要升变的兵所属的玩家编号。

~PromotionDialog();

析构函数

char getValue();

返回 val，即选择的结果。参看 val。

private slots:

void b1_clicked();

连接到第一个按钮的 clicked()信号，将 val 设置为'Q'或'q'。参看 val。

void b2_clicked();

void b3_clicked();

void b4_clicked();

参看 b1_clicked()，第 2~4 个按钮分别设置 val 为：'R'或'r','N'或'n','B'或'b'。

private:

Ui::PromotionDialog *ui;

存放构造中实例化的 ui 对象指针。

```
int m_player;
```

玩家编号。

```
char val;
```

选择的结果，默认为'Q'或'q'。