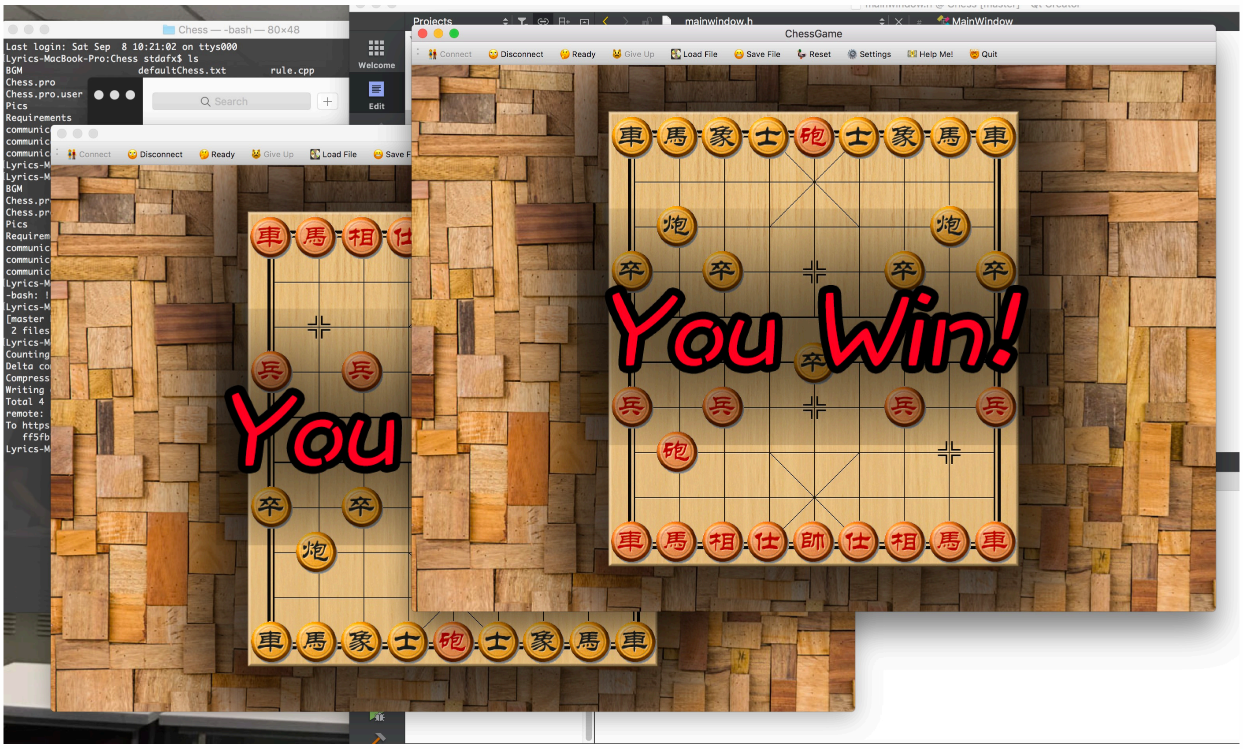
Happy Qt Chinese Chess

Introdution

- A Qt-based online Chinese chess game platform.
- Developed by LyricZ, 2017011362.
- A summer programming training project, 2018 Summer.
- Github: https://github.com/LyricZhao/QtChineseChess



```
Last login: Sat Sep  8 10:21:02 on ttys000
Lyrics-MacBook-Pro:Chess stdafx$ ls
BGM                     defaultChess.txt        rule.cpp
Chess.pro
Chess.pro.user
Pics
Requirements
communic
communic
communic
Lyrics-M
Lyrics-M
BGM
Chess.pr
Chess.pr
Pics
Requirem
communic
communic
communic
Lyrics-M
-bash: !
Lyrics-M
[master
 2 files
Lyrics-M
Counting
Delta co
Compress
Writing
Total 4
remote:
To https
   ff5fb
Lyrics-M
```

Features

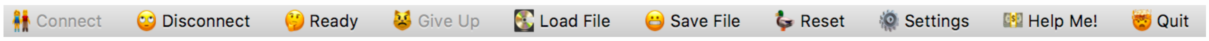
- √ OOP Design Style, Only 1.2k LoC.
- √ High Scalability
- √ High-Efficient Implement
- √ Cross-Platform
- √ Colorful Display
- √ Sound Effect
- √ Smooth User Experience
- √ File Save/Load
- √ Comfortable Coding Style
- √ Tcp High Stability

Environments

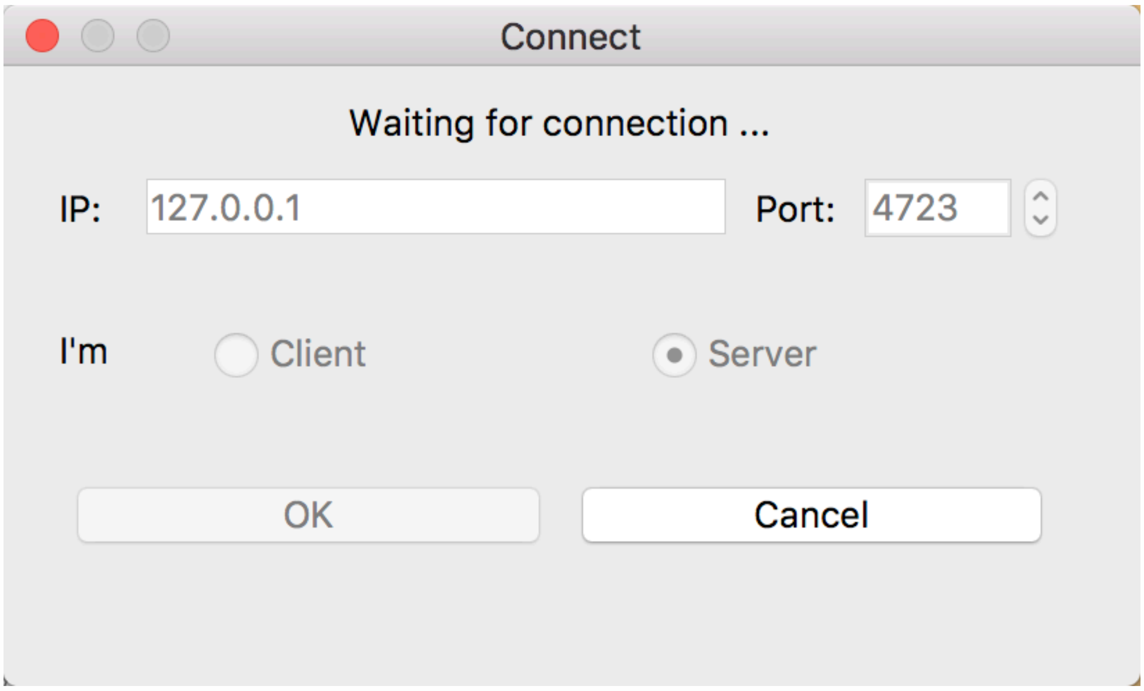
- Local Compiling Environment
  - macOS 10.13.4
  - Qt 5.10.1
  - Clang 902.0.39.1
- I don't compile the code on Windows or Linux, I am not sure whether there will be some problems.

How to game

- About the game, there are 3 global status of the system:
  - Not connected, you can do nothing but to connect.
  - Pending, load file or get ready for the game.
  - Gaming, enjoy and focus on your game!
- ToolBar



  - Connect
    - You can choose to be the client or the server.
    - When you choose to be client, you should input the IP address and the port of the server, if server, you just should input the port you want to listen.
    - In fact, the client and server are equal but only that the server can listen and wait for the client to connect.



Connect

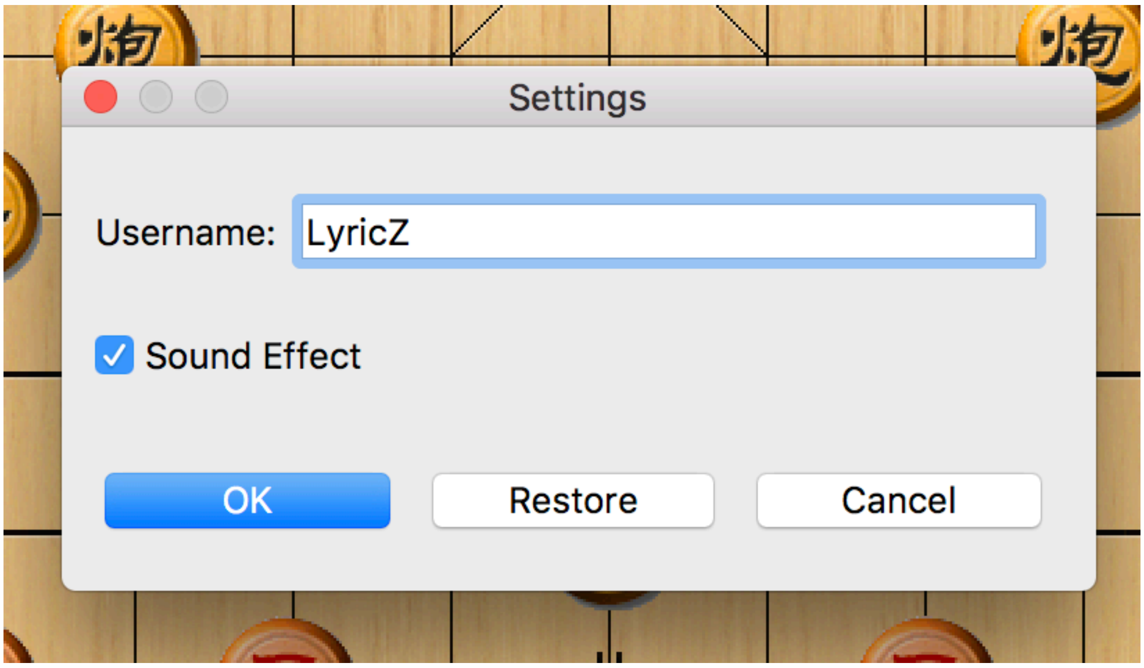
Waiting for connection ...

IP: 127.0.0.1 Port: 4723

I'm Client [✓ Server]

  - Disconnect
    - When you are in the status of Pending or Gaming, you can click it and disconnect the current connection.
    - When you click, you will quit the current status to Not connected status, but the status of chessboard will be saved and you can save it into a file.
  - Ready

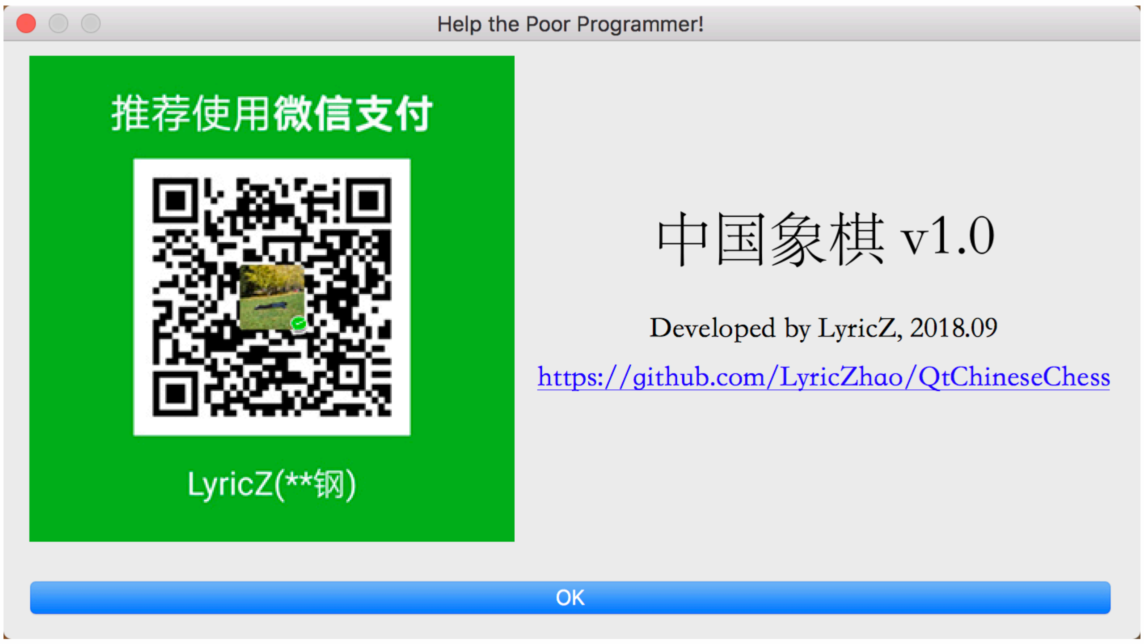
    - This function can only be used in Pending mode, click it and it means that you are satified with the chessboard and ready for the game, the status of the players will be displayed.
    - Click again to cancel the ready status.
    - When two players are all ready for the game, the game will start.
    - The player which will move first is the one who gets ready first.
- Give Up
    - This function is only available in Gaming mode.
    - Clicking this means that you surrender and quit the Gaming status and back to the Pending status.
- Load file
    - Load a chessboard file from the disk when you are not gaming.
    - If you are in the Pending mode and load a file, your board will be exactly set to the file and another player will be the another side of the status as the file describes.
    - It is very fair because the game can only be launched when the two both accept and are ready for the game.
- Save file
    - You can do it anytime!
    - The file will be saved into the path you choose.
    - Note that the timer (when gaming) will not be stopped when you are saving a file.
- Reset
    - You can reset the chessboard back to the default chessboard by just clicking it in Not connected or Pending mode.
    - In fact, to simplify the code, the Reset button is just an implement of loading file of the default status.
- Settings
    - You can change the settings anytime!
    - The username(in fact no use..) and the sound effect can be set here.
    - Note that the timer (when gaming) will not be stopped when you are setting.


Settings

Username: LyricZ

✓ Sound Effect

- Help me
    - Help the poor programmer!



中国象棋 v1.0

Developed by LyricZ, 2018.09

https://github.com/LyricZhao/QtChineseChess

  - Quit
    - Quit the game anytime!
- Note that the timer is limited to 20s, which means that you need to move in 20s for a single step.

About the Code

- I think the code is pretty OOP-Style and comfortable to read, all functions are implemented in only 1.2k LoC.
- The complex rules in Chinese chess are implemented in only about 1h LoC using a lots of macros and elegant expressions, e.g. (The implement of getting the reaching points of a chariot)

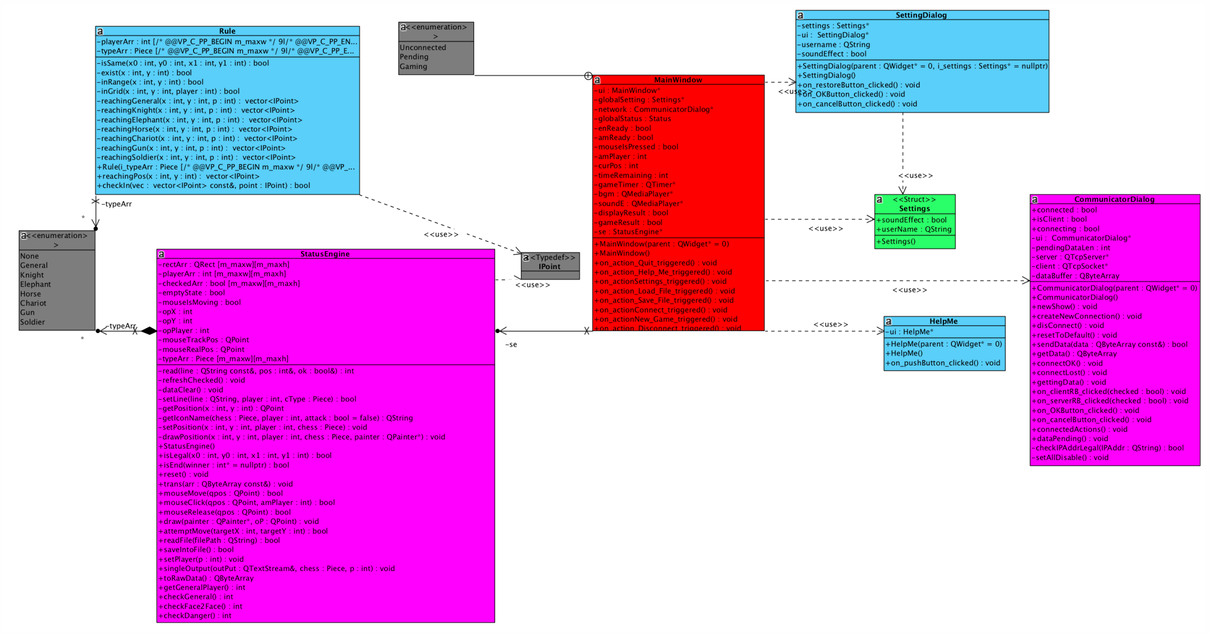
```
# define funcChariot if(!inRange(rx, ry) || isSame(x, y, rx, ry)) break;
\
    ret.push_back(MP(rx, ry)); \
    if(exist(rx, ry)) break; \

std:: vector<IPoint> Rule:: reachingChariot(int x, int y, int p) {
    (void)(p);
    std:: vector<IPoint> ret; ret.clear();
    for(int rx = x - 1, ry = y; ; -- rx) { funcChariot; }
    for(int rx = x + 1, ry = y; ; ++ rx) { funcChariot; }
    for(int rx = x, ry = y - 1; ; -- ry) { funcChariot; }
    for(int rx = x, ry = y + 1; ; ++ ry) { funcChariot; }
    return ret;
}
```

- Also, there are some lambda expression in C++ 11 to simplify the code.

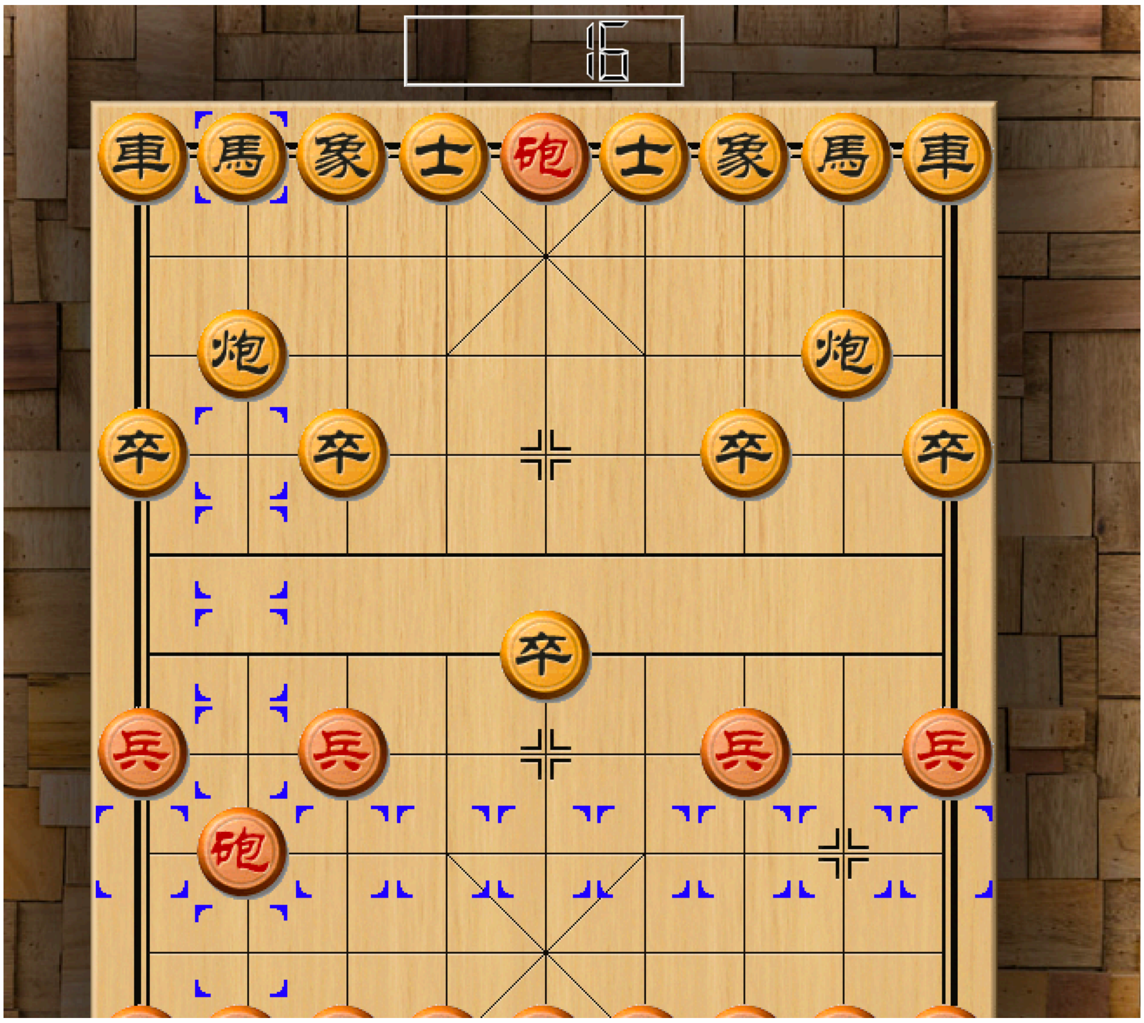
```
connect(client, &QTcpSocket:: disconnected, this, [this](){ emit
connectLost(); });
```

- Almost all the functions are controlled in 10 lines and all the key components are packaged in a class ensuring the scalability.
- The relationships of the components are as below:


Rule
-playerArr : int [/* @@VP_C_PP_BEGIN m_maxw */ 9]/* @@VP_C_PP_EN...
-typeArr : Piece [/* @@VP_C_PP_BEGIN m_maxw */ 9]/* @@VP_C_PP_E...
-isSame(x0 : int, y0 : int, x1 : int, y1 : int) : bool
-exist(x : int, y : int) : bool
-inRange(x : int, y : int) : bool
-inGrid(x : int, y : int, player : int) : bool
-reachingGeneral(x : int, y : int, p : int) : vector<IPoint>
-reachingKnight(x : int, y : int, p : int) : vector<IPoint>
-reachingElephant(x : int, y : int, p : int) : vector<IPoint>
-reachingHorse(x : int, y : int, p : int) : vector<IPoint>
-reachingChariot(x : int, y : int, p : int) : vector<IPoint>
-reachingGun(x : int, y : int, p : int) : vector<IPoint>
-reachingSoldier(x : int, y : int, p : int) : vector<IPoint>
+Rule(i_typeArr : Piece [/* @@VP_C_PP_BEGIN m_maxw */ 9]/* @@VP_...
+reachingPos(x : int, y : int) : vector<IPoint>
+checkIn(vec : vector<IPoint> const&, point : IPoint) : bool
-typeArr
*
<<enumeration>>
Unconnected
Pending
Gaming
<<enumeration>>
None
General
Knight
Elephant
Horse
Chariot
Gun
Soldier
-typeArr
*
<<use>>
<<Typedef>>
IPoint
<<use>>
-se
StatusEngine
-rectArr : QRect [m_maxw][m_maxh]
-playerArr : int [m_maxw][m_maxh]
-checkedArr : bool [m_maxw][m_maxh]
-emptyState : bool
-mouseIsMoving : bool
-opX : int
-opY : int
-opPlayer : int
-mouseTrackPos : QPoint
-mouseRealPos : QPoint
-typeArr : Piece [m_maxw][m_maxh]
-read(line : QString const&, pos : int&, ok : bool&) : int
-refreshChecked() : void
-dataClear() : void
-setLine(line : QString, player : int, cType : Piece) : bool
-getPosition(x : int, y : int) : QPoint
-getIconName(chess : Piece, player : int, attack : bool = false) : QString
-setPosition(x : int, y : int, player : int, chess : Piece) : void
-drawPosition(x : int, y : int, player : int, chess : Piece, painter : QPainter*) : void
+StatusEngine()
+isLegal(x0 : int, y0 : int, x1 : int, y1 : int) : bool
+isEnd(winner : int* = nullptr) : bool
+reset() : void
+trans(arr : QByteArray const&) : void
+mouseMove(qpos : QPoint) : bool
+mouseClick(qpos : QPoint, amPlayer : int) : bool
+mouseRelease(qpos : QPoint) : bool
+draw(painter : QPainter*, oP : QPoint) : void
+attemptMove(targetX : int, targetY : int) : bool
+readFile(filePath : QString) : bool
+saveIntoFile() : bool
+setPlayer(p : int) : void
+singleOutput(outPut : QTextStream&, chess : Piece, p : int) : void
+toRawData() : QByteArray
+getGeneralPlayer() : int
+checkGeneral() : int
+checkFace2Face() : int
+checkDanger() : int
MainWindow
-ui : MainWindow*
-globalSetting : Settings*
-network : CommunicatorDialog*
-globalStatus : Status
-enReady : bool
-amReady : bool
-mouseIsPressed : bool
-amPlayer : int
-curPos : int
-timeRemaining : int
-gameTimer : QTimer*
-bgm : QMediaPlayer*
-soundE : QMediaPlayer*
-displayResult : bool
-gameResult : bool
-se : StatusEngine*
+MainWindow(parent : QWidget* = 0)
+MainWindow()
+on_action_Quit_triggered() : void
+on_action_Help_Me_triggered() : void
+on_actionSettings_triggered() : void
+on_action_Load_File_triggered() : void
+on_action_Save_File_triggered() : void
+on_actionConnect_triggered() : void
+on_actionNew_Game_triggered() : void
+on_action_Disconnect_triggered() : void
<<use>>
SettingDialog
-settings : Settings*
-ui : SettingDialog*
-username : QString
-soundEffect : bool
+SettingDialog(parent : QWidget* = 0, i_settings : Settings* = nullptr)
+SettingDialog()
+on_restoreButton_clicked() : void
+on_OKButton_clicked() : void
+on_cancelButton_clicked() : void
<<use>>
<<Struct>>
Settings
+soundEffect : bool
+userName : QString
+Settings()
<<use>>
<<use>>
<<use>>
HelpMe
-ui : HelpMe*
+HelpMe(parent : QWidget* = 0)
+HelpMe()
+on_pushButton_clicked() : void
CommunicatorDialog
+connected : bool
+isClient : bool
+connecting : bool
-ui : CommunicatorDialog*
-pendingDataLen : int
-server : QTcpServer*
-client : QTcpSocket*
-dataBuffer : QByteArray
+CommunicatorDialog(parent : QWidget* = 0)
+CommunicatorDialog()
+newShow() : void
+createNewConnection() : void
+disConnect() : void
+resetToDefault() : void
+sendData(data : QByteArray const&) : bool
+getData() : QByteArray
+connectOK() : void
+connectLost() : void
+gettingData() : void
+on_clientRB_clicked(checked : bool) : void
+on_serverRB_clicked(checked : bool) : void
+on_OKButton_clicked() : void
+on_cancelButton_clicked() : void
+connectedActions() : void
+dataPending() : void
-checkIPAddrLegal(IPAddr : QString) : bool
-setAllDisable() : void

- Here are the details of the design:
  - communicator.cpp/h
    - This file is about the network connection and the send/receive data functions are also implemented in the file.
    - The implement uses QTcpSocket and QTcpServer, the behaviors of the server and client will be exactly same after they establish a connection and it's esay for debugging.
    - Here I also consider the situtation that message is splitted or sticky to one, and also implement the error handler when losting connection.
    - The file also implement the connection dialog and can check the correctness of the IP address.
  - helpme.cpp/h
    - The implement of the "Help me" dialog.
    - Pretty interesting.
  - rule.cpp/h
    - A filter which can return the reaching points of a selected chess piece.
    - You can create one with the chessboard status and input the position, the class Rule will return a std:: vector showing where the piece can go.
  - setting.cpp/h
    - The setting dialog implement.
  - statusengine.cpp/h
    - Almost the core of the game.
    - The class StatusEngine save the status of the game and is responsible for drawing the chessboard.
    - Also, the file saving/loading functions are also implemented in the class.
  - mainwindows.cpp/h
    - We can say that the class MainWindow is a bridge for all the components and connect them into a completed game.

- The code is not very complex because the main work is finished in the sub-working classes.
- The most important work of the main window is handling the events such as the paintEvent or MouseMove, etc..
- What I want to highlight is that the movement of the chess piece can be finished by dragging the chess and the target legal places to move will be highlighted which is not very easy to code.



How does it work?

- When the game starts, only StatusEngine will be initiated, this part is responsible for drawing the chessboard and save the status of the board, and MainWindow is always running to call these functions and do some scheduling works.
- About the network communicator:
  - The network communicator is package in a new class, handling sending and receiving data.
  - When it starts, it will start a TcpServer or TcpSocket, then the two players will establish the connection.
  - For transfering data, the communicator will send a header of the message size first, and the other client will wait all the data to reach, and then a signal is emitted.
  - The message is designed in a format of [size:type:context], the [size] is 4 bytes, the same with [type], the [size] means the size of the entire message, and for different types:

        - type = 0: all the data of the chessboard, the context includes the player and the type of each position.
        - type = 1: it means I(the sender) am ready for the game.
        - type = 2: it means I(the sender) am lost.
    - The next state of the global game is judged locally for programming-friendly, and also for the equality of the two players.
- The transfer of data of Communicator and the MainWindow is by signals and slots, for MainWindow and the StatusEngine is just by calling functions.

About the Interface and Sound Effect

- The icons of the pieces and chessboard are downloaded from the Internet.
- The background is also downloaded from the Internet, but the shadow effect of the board is made by myself using PhotoShop, I spent a lot of time on the details.
- The sound effect is extracted from QQ Dou Di Zhu, that why I call the game "Happy" Chess, I think building a game like it is pretty interesting (although a little bothering about the complex code structure)

Thanks

- Thanks to my roommate and TA to help me test the game!

Enjoy the game!