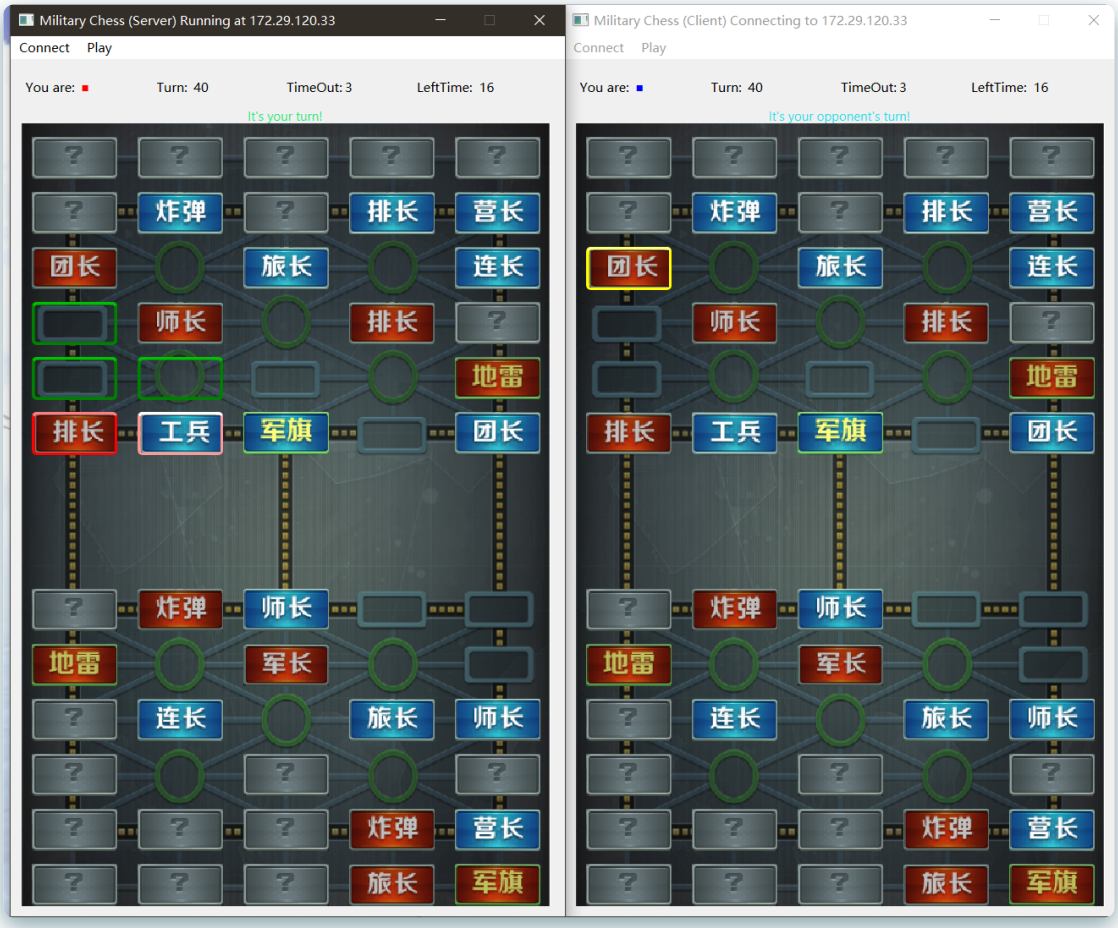
✓ Military Chess

Homework 1 for 2020-2021 Summer Course, CST.



- For information about the developer

写于 author.txt 内，已添加入 .gitignore 文件中.

用于网络学堂身份验证.

- For information about this repo

Would be made public after the final submission date of the homework and grading process.

If you are a student taking this course years later, please **DO NOT** copy any of these files directly into your homework. The developer will not be responsible for any possible loss.

✓ 报文及进程间通信机理

Server ⟷ Client（进程间）

Pre-game messages 预处理报文

`200` Connection Established

当 Client 连接到 Server 之后：

- Server 向 Client 发送 `200` 信号，Client 接收后返回 `200` 信号，完成第一次通信
- Server 在收到 `200` 信号后，会生成棋盘内容，然后发送 `201` 信号，将棋盘数据传输给客户端

`201` Initialize Chessboard

- 使用格式：`201 <Chessboard Data>`，`201 Received`
- 服务端使用 `201` 信号来传输数据，客户端收到 `201` 信号后返回 `201` 信号，此时两个进程同时进入 `WAIT_PLAY_CONFIRMATION` 状态

`202` Client Ready

- 两者都准备就绪是由服务端来判断的，因此客户端点击 `Start` 按钮后，向服务端发送报文 `202` 以表示自己准备完毕
- 此时服务端可以在客户端之前或之后完成准备过程

`203` Game Start

- 使用格式 `203 <Offensive>` ，其中 0 代表服务端先手，1 代表客户端先手
- 一旦 Server 自身完成准备，也接受到了 `202` 信号，便会生成一个介于 0 和 1 之间的随机数，然后向客户端发送该报文
- 服务端与客户端同时由 `WAIT_PLAY_CONFIRMATION` 状态转换为 `PLAYING_WAITING` 或 `PLAYING_THINKING` 状态，游戏开始

In-game messages 数据操作报文

`300` Reveal Chess Piece

- 使用格式：`300 <position>`
- 将会把位于 position 位置的棋子翻开

`301` Move Chess Piece

- 使用格式：`301 <from> <to>`
- 将会把位于 from 位置的棋子移动到 to 位置

`302` Eat Chess Piece

- 使用格式：`302 <from> <to>`
- 将会把位于 to 位置的棋子标记为死亡，然后把 from 位置的棋子移动到 to 位置

`303` Eat Chess Piece and Destroy Self

- 使用格式：`303 <from> <to>`
- 将会把位于 `from, to` 位置的棋子标记为死亡

Switch-control messages 控制权交换报文

`400` Switch Turn

- 一个**回合**结束后的进程将发送 `400` 报文
- 接收到 `400` 报文的机器将会开始一个新的**回合**

`401` Win Game

- 使用格式：`401 <isMe>` 其中 `1` 代表自方胜利，`0` 代表对方胜利
- 使用后游戏转入 `End` 状态

Game ⟷ GameConnection（多线程间）

除了上述报文之外，为方便进行提示文字的更改，在线程间通信时还增加了以下规则：

`0 <Color> <Message>`

- 说明：设置主界面的说明文字
- 参数
  - `Color`: 报文的颜色(六位十六进制数，或是HTML支持的颜色标签)，如`ff0000`
  - `Message`: 说明文字.

✓ 常量设计与主要算法

常量设计

ChesePiece

棋子的 `InitID` 与其兵种的对应关系如下。其中偶数为蓝方，奇数为红方。

InitID	兵种
1~2	军旗
3~4	司令
5~6	军长
7~10	师长
11~14	旅长
15~18	团长

InitID	兵种
19~22	营长
23~26	炸弹
27~32	连长
33~38	排长
39~44	工兵
45~50	地雷

棋子类均从抽象基类 `ChessPiece` 类派生，并重载了其中的 `getArmType()` 方法和 `canEat(ChessPiece*)` 方法（获取兵种，是否可吃另一种棋子）。

~~虽然也想到了使用 Rank 来控制棋子行为的想法，但这样很不 OOP，况且用 Python 序列化地生成出 C++ 代码类来也并不困难~~

Enumerations

为代码的可读性考虑，创建了 `ArmType`，`Faction`，`EnumChessPiece` 等等枚举类.

Images

图片使用静态方法在 MainWindow 刚刚载入时便加载进了内存中，以便后续的调用.

Graph

- 60 个顶点按照 Row Major 顺序映射为了 [0, 60) 间的整数.
- 边使用 `QVector<Edge>` 来存储，其中 `typedef Edge QVector<int>({int from, int to, int type})`，保证 `from < to`，且 `type` 值 1 为公路， 2 为铁路.
- 行营额外使用了 `QVector<int>` 来存储.

GameLogic 类

整个 GameLogic 类由静态成员和静态方法构成，为 Game 类的友元类，便于调用以控制游戏的流程逻辑和规则逻辑.

主要算法

深度优先搜索

在判断工兵的可移动范围时，使用了深度优先搜索算法.

✓ 网络通信编程框架

总体上通信采用了 `QTcpServer` 和 `QTcpSocket` 两个类，多线程采用了 `QThread` 类.

为防止主线程的阻塞，每个进程会为通信单独开启一个子线程.

两个主进程之间的数据交流是通过两个通信子线程之间的通信来实现的.

而主线程与通信线程之间则通过信号槽机制来进行数据的传输.

Reference:

- https://zhuanlan.zhihu.com/p/53270619
- https://blog.csdn.net/bailang_zhizun/article/details/78327974

✓ 信号槽机制的设计

涉及到信号槽机制的地方十分普遍，这里我们归类进行说明.

主界面与弹出窗口

Menubar 主界面菜单栏

- 5 个菜单选项的点击信号及其对应的槽
- 接收 Game 类发送的信号控制菜单选项的开启与否

Event Listener 主界面点击事件

主界面上的 60 个 QLabel 和 1 个底层 QWidget 被注册了点击事件的过滤器，用于点击事件的处理.

弹出窗口内部的数据流

主要用于客户端连接服务端时输入 IP 地址的小键盘（使用 Lambda 函数）与文本输入框之间的数据交流.

弹出窗口与主界面的数据传输

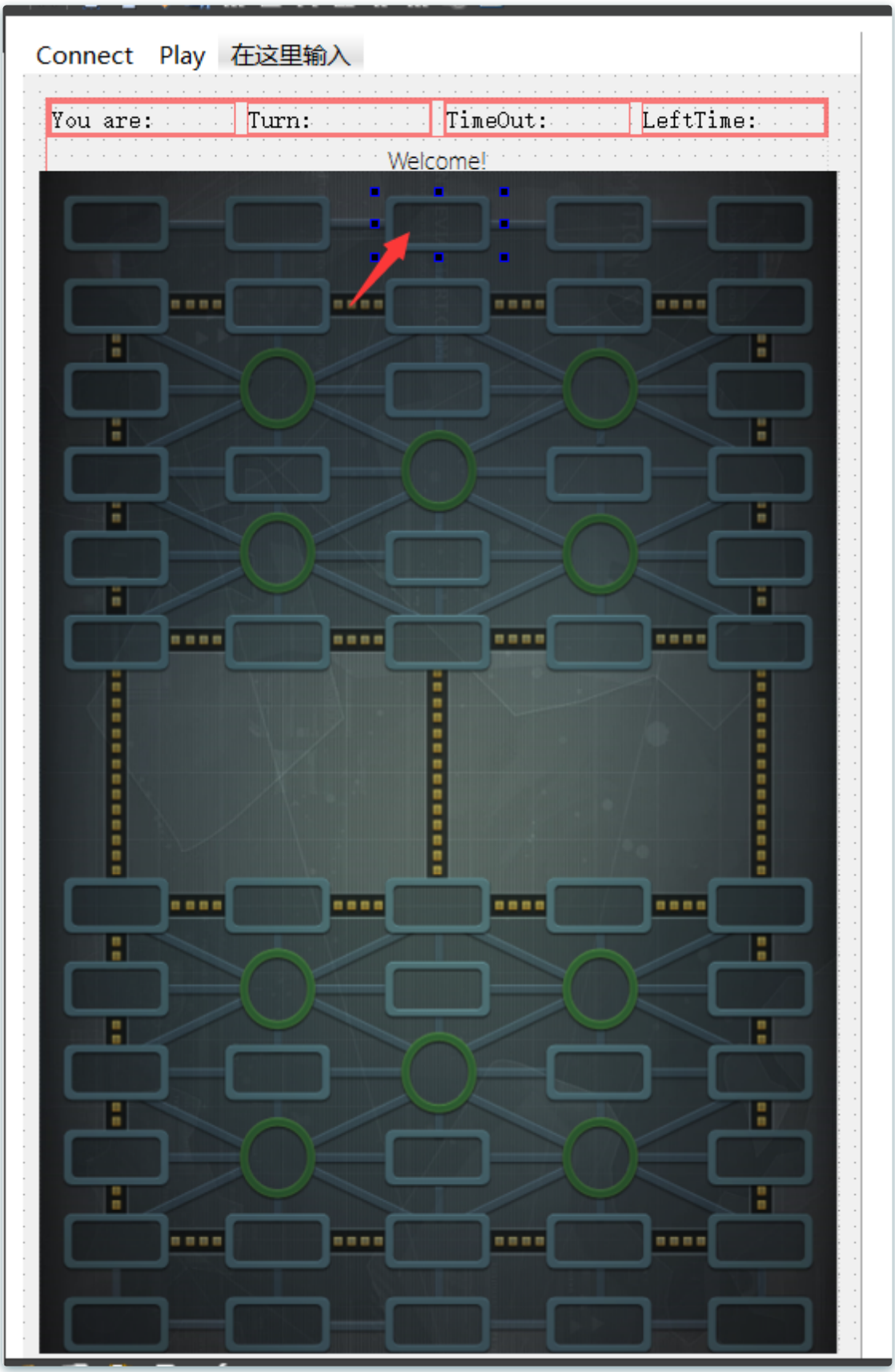
用于点击 OK 按钮后，将 IP 数据以信号的形式发送给主界面.

Game 类与 GameConnection 类

- 建立连接，收发数据与取消连接的信号槽
- 设定主界面显示数据的信号
- 接收主界面点击信号来处理的槽函数
- 定时器定时触发的槽函数

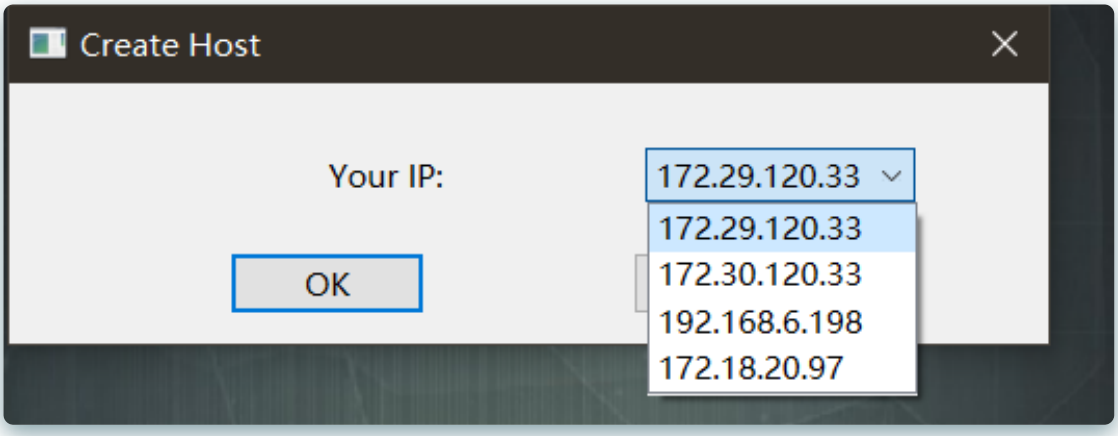
✓ GUI 界面设计

主界面



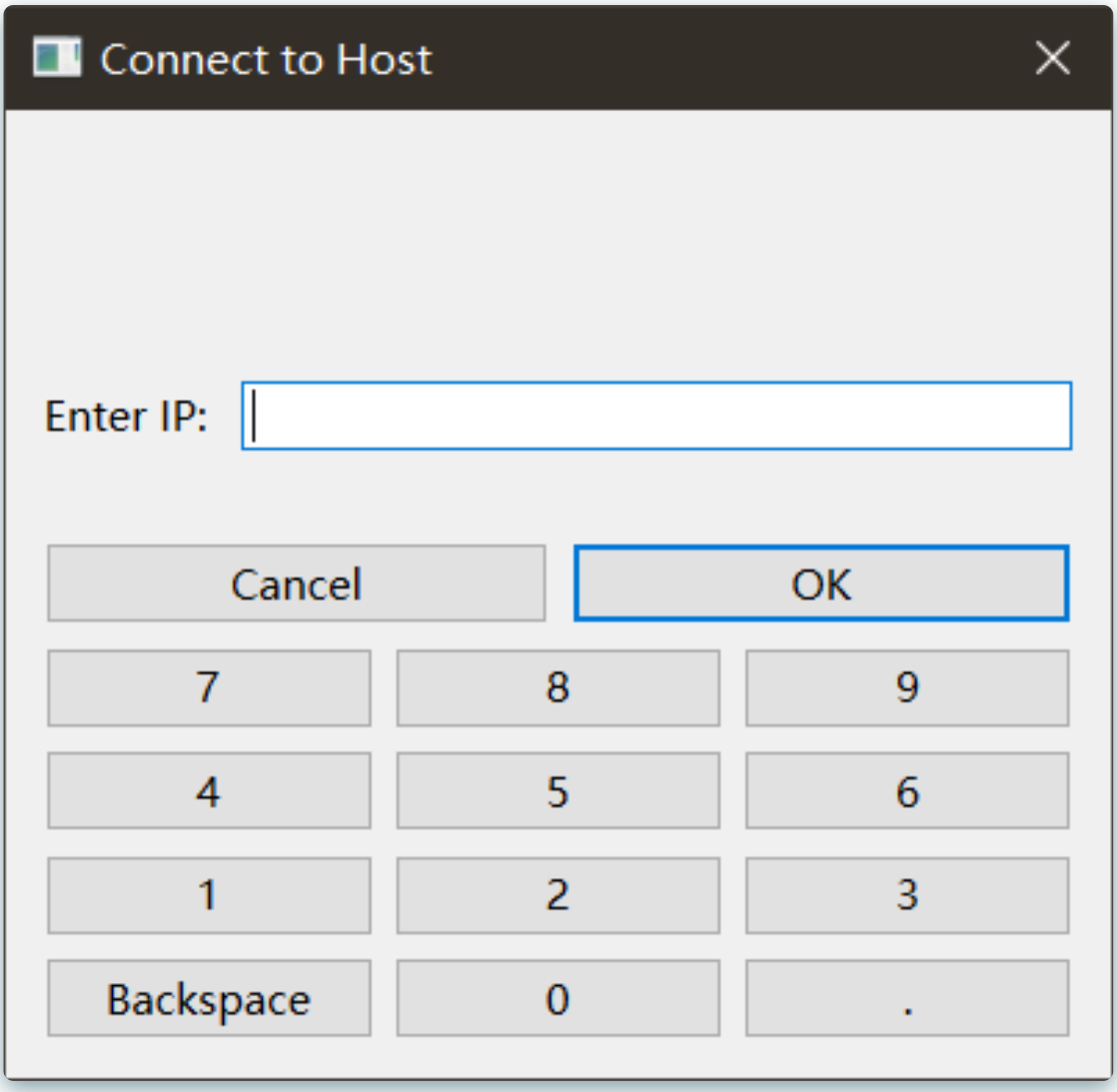
- 上方：菜单栏，游戏状态信息，提示信息
- 下方
  - 地图背景
  - 一个 QWidget，内含 60 个 QLabel

创建主机



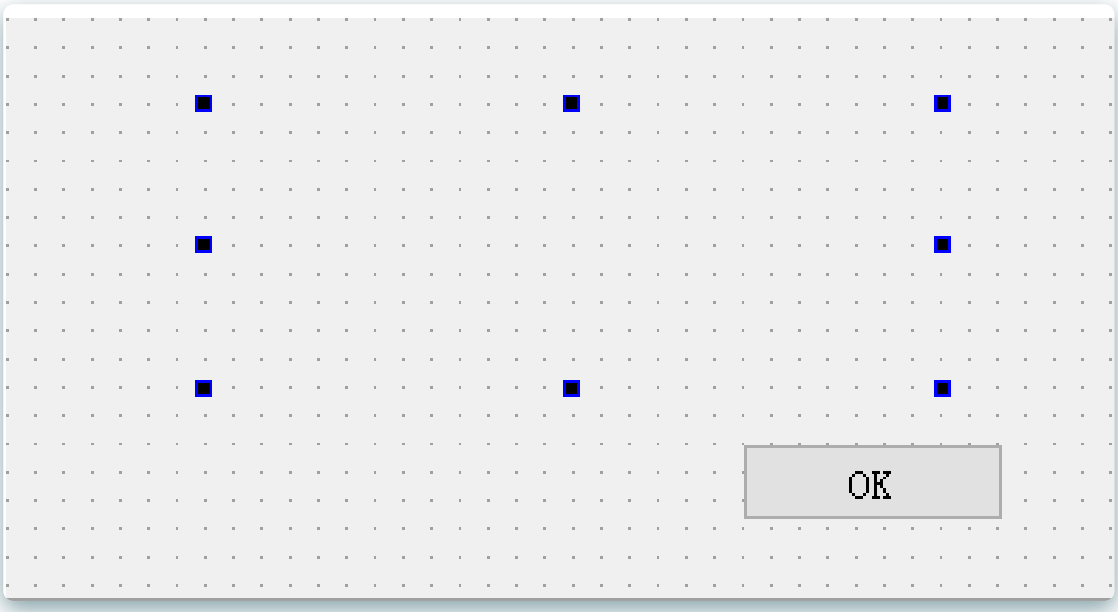
使用下拉框选择 IP.

连接到主机



实现了输入 IP 地址的正则匹配.

游戏结束界面



✓ 可能出现的已知问题

游戏窗口失焦后，在进行棋子移动时有极小概率可能在原地会留下白框

初步排除了代码逻辑错误导致的问题，且该现象在游戏窗口再次成为焦点后自动解除，推断疑似为 Qt 未能为该窗口自动进行重新渲染.

为什么界面那么丑?

First edited by c7w, 2021-8-19 22:18:46.