基于Qt的带有危险提醒功能的网络联机五子棋对战游戏软件

本项目实现了一个基于Qt的可以双人联网对战的五子棋游戏，在特定情况下，还会给出危险提示。这些情况包括：对方再落一子，出现至少两个无阻挡的三个连续子；对方再落一子，出现至少一个无阻挡的三个连续子以及一个有单侧阻挡的四个连续子；对方再落一子，出现至少一个无阻挡的四个连续子；对方再落一子，获胜。

总体架构

本项目分为主机程序和客户端程序。其中，客户端程序只包括客户端模块，主机程序包括客户端模块以及服务器模块。

客户端模块

客户端模块主要由`widget.cpp`实现，包括一个自制的`QWidget`的子类实例用于绘制棋盘（以及上面的棋子、炸弹等）以及一些`QLabel`来显示信息，它负责接收服务器的指令来进行显示、锁定、解锁棋盘以及显示信息等操作，并根据用户操作向服务器发送相关操作数据来进行游戏，还进行一些不关键的计算，比如危险提醒。用户可以直接操作客户端模块来连接服务器，进而开始游戏。

服务端模块

服务端模块主要由`chessserver.cpp`实现，负责提供服务、与客户端模块通信。服务端模块可以接收客户端模块发来的请求，更新棋盘、开始游戏。服务器模块还进行关键的计算，包括获胜的计算、玩家落子的计算（检查是否轮到他，目标位置是否可放子等），来防止客户端作弊，保证游戏的公平性。用户可以利用服务端模块创建游戏服务器，两个客户端模块连接之后，便可开始游戏。

顺便指出，由于本架构的特点，它可以轻易地升级为有中心服务器的支持多人两两对战的五子棋游戏平台。中心服务器只需要加载服务器模块来进行计算和服务，客户端甚至不需要太大改动。

通信协议

通信，指的是客户端模块与服务器模块的通信。

客户端模块与服务器模块通信方式分为本地通信以及网络通信两种，但从效果来看，它们传输的数据是相同的，均为`QJsonObject`对象。

本地通信

一种是直接使用Qt的信号槽机制来传输`QJsonObject`对象。这种方法适用于服务端模块和客户端模块处于同一进程，包括同一线程和不同线程的情况。

网络通信

一种是使用Qt的Socket网络编程框架的`QTcpSocket`类以及一个自制的用于处理协议的`JsonSession`实例与服务器通信。

`JsonSession`在每次`QTcpSocket`的`readyRead()`信号触发时读取内容，记录当前读取状态以便之后继续读取未读取完毕的数据包和新到达的数据包，当一个包接收完毕之后，触发一个`onMessage(QJsonObject)`信号来宣告收到数据，通知相关模块进行处理。发送的时候，直接将`QJsonObject`对象根据协议发送到`QTcpSocket`，传输到网络。这种方法适用于服务端模块和客户端模块通过网络连接的情况。

具体的，数据包使用TCP在网络上传输，每个数据包由四个魔术字符`GMKU`开头，后面紧跟数据部分的长度（大端序表示）。数据部分就是`QJsonObject`对象序列化之后得到的JSON字符串，通常以`{`开头，`}`结尾。

```
 0       3 4       7 8    ...
+--------+---------+----...
| G M K U|数据长度|数据...
+--------+---------+----...
```

接着，下面介绍数据包中传输的`QJsonObject`对象中的信息。

`QJsonObject`对象中存储了两个字段，`type`表示指令/请求的类型，`data`中存储了具体的数据，可能为`QJsonObject`或者基本数据类型。

指令/请求集

下面，以`type`字段作为标题。服务器指的是服务器模块，客户端指的是客户端模块。指令是服务器发给客户端的，请求是客户端发给服务器的。

hello（指令/请求）

服务器和客户端互相打招呼，预示着新一轮游戏马上开局。

new（请求）

服务器在连接确认之后会自动开局，如果一个回合结束之后还需要开局，则客户端需要发送`new`指令到服务器。

无`data`字段。

color（指令）

颜色分配指令。

`data`字段是一个只有一个字符的字符串，存储了客户端这一局的颜色信息。可能的取值为`"W"`、`"B"`。

update（指令）

更新棋盘指令。

`data`字段是一个JSON，包括了如下字段。

- `row`，整数，最后放置的棋子的行坐标，如果没有棋子，那么为`-1`
- `col`，整数，最后放置的棋子的列坐标，如果没有棋子，那么为`-1`
- `turn`，类型同`color`的`data`，当前可以放置棋子的玩家的颜色。
- （可选的）`board`，是一个二维数组，元素类型同`color`的`data`，第一维下标指示行坐标，第二维下标指示列坐标。如果含有此字段则说明是全棋盘刷新，否则只根据`row`、`col`来增量更新棋盘。

place（请求）

放置棋子。

`data`字段是一个JSON，包括了如下字段。

- `row`，整数，要放置的棋子的行坐标
- `col`，整数，要放置的棋子的列坐标

注意：服务器会进行落子的计算（检查是否轮到该颜色，目标位置是否可放子等），通过后再更新棋盘，向双方发送`update`指令并计算是否出现获胜或平局的情况，按计算结果发送`win`指令。

win（指令）

获胜信息。

`data`字段类型同`color`的`data`，存储了这一局获胜的颜色。除了`color`的`data`中描述的取值之外，还包括`"-"`表示平局。

工作流程

下面，结合通信协议配合一些例子来说明主机程序和客户端程序的工作流程。

例子

下面的例子中传输的数据由真实TCP流抓包获得。`S`表示由服务器发出的信息，`C`则表示是客户端发出的。数据中的`.`表示一个不可打印的字符。

一轮回合

首先，在主机程序创建一个服务器，然后使用客户端连接。

连接之后客户端会发送

```
C: GMKU....{"type":"hello"}
```

之后，服务器自动开局，发送客户端的颜色，当前棋盘等信息。

```
S: GMKU....{"type":"hello"}
S: GMKU....{"data":"B","type":"color"}
S: GMKU....{"data":{"board":[[" "," "," "," "," "," "," "," "," "," "," "," "," "
","," "," "," "],[" "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "],
[" "," "," "," "," "," "," "," "," "," "," "," "," "," "," "],[" "," "," ","
","," "," "," "," "," "," "," "," "," "," "," "],[" "," "," "," "," "," "," ","
","," "," "," "," "," "," "," "],[" "," "," "," "," "," "," "," "," "," "," ","
","," "," "," "],[" "," "," "," "," "," "," "," "," "," "," "," "," "," "," "],
[" "," "," "," "," "," "," "," "," "," "," "," "," "," "," "],[" "," "," ","
","," "," "," "," "," "," "," "," "," "," "," "],[" "," "," "," "," "," "," ","
","," "," "," "," "," "," "," "],[" "," "," "," "," "," "," "," "," "," "," ","
","," "," "],[" "," "," "," "," "," "," "," "," "," "," "," "," "," "," "],
[" "," "," "," "," "," "," "," "," "," "," "," "," "," "," "],[" "," "," ","
","," "," "," "," "," "," "," "," "," "," "," "],[" "," "," "," "," "," "," ","
","," "," "," "," "," "," "
"]],"col":-1,"row":-1,"turn":"B"},"type":"update"}
```

然后，客户端程序发出落子请求。

```
C: GMKU...){"data":{"col":7,"row":7},"type":"place"}
```

服务器在判断之后，确认落子成功。

```
S: GMKU...5{"data":{"col":7,"row":7,"turn":"W"},"type":"update"}
```

接着，另外一个客户端程序落子，服务器判断并确认。

```
(对方落子)
S: GMKU...5{"data":{"col":8,"row":6,"turn":"B"},"type":"update"}
```

就这样，多次落子，直到出现输赢。

```
C: GMKU...){"data":{"col":9,"row":6},"type":"place"}
S: GMKU...5{"data":{"col":9,"row":6,"turn":"W"},"type":"update"}
(对方落子)
S: GMKU...5{"data":{"col":8,"row":7,"turn":"B"},"type":"update"}
...(轮流落子)...
C: GMKU...){"data":{"col":6,"row":2},"type":"place"}
S: GMKU...5{"data":{"col":6,"row":2,"turn":"W"},"type":"update"}
(对方落子)
S: GMKU...5{"data":{"col":6,"row":1,"turn":"B"},"type":"update"}
C: GMKU...){"data":{"col":6,"row":6},"type":"place"}
S: GMKU...5{"data":{"col":6,"row":6,"turn":"W"},"type":"update"}
```

这个时候，服务端检测到了黑棋赢，发送了下面的消息。

```
S: GMKU....{"data":"B","type":"win"}
```

客户端断开了连接，游戏结束。

多次回合

第一局的流程是一样的。

```
C: GMKU....{"type":"hello"}
S: GMKU....{"type":"hello"}
S: GMKU....{"data":"B","type":"color"}
S: GMKU....{"data":{"board":[[" "," "," "," "," "," "," "," "," "," "," "," "
","," "," "," "],[" "," "," "," "," "," "," "," "," "," "," "," "," "," "," "],
[" "," "," "," "," "," "," "," "," "," "," "," "," "," "," "],[" "," "," "
","," "," "," "," "," "," "," "," "," "," "," "],[" "," "," "," "," "," "," ","
","," "," "," "," "," "," "," "],[" "," "," "," "," "," "," "," "," "," "," "
","," "," "," "],[" "," "," "," "," "," "," "," "," "," "," "," "," "," "," "],
[" "," "," "," "," "," "," "," "," "," "," "," "," "," "," "],[" "," "," "
","," "," "," "," "," "," "," "," "," "," "," "],[" "," "," "," "," "," "," ","
","," "," "," "," "," "," "
"]],"col":-1,"row":-1,"turn":"B"},"type":"update"}
C: GMKU...){"data":{"col":7,"row":7},"type":"place"}
S: GMKU...5{"data":{"col":7,"row":7,"turn":"W"},"type":"update"}
(对方落子)
S: GMKU...5{"data":{"col":8,"row":5,"turn":"B"},"type":"update"}
C: GMKU...){"data":{"col":7,"row":5},"type":"place"}
S: GMKU...5{"data":{"col":7,"row":5,"turn":"W"},"type":"update"}
(对方落子)
S: GMKU...5{"data":{"col":7,"row":6,"turn":"B"},"type":"update"}
C: GMKU...){"data":{"col":9,"row":4},"type":"place"}GMKU...5{"data":
...(轮流落子)...
C: GMKU...){"data":{"col":5,"row":3},"type":"place"}
S: GMKU...5{"data":{"col":5,"row":3,"turn":"W"},"type":"update"}
(对方落子)
S: GMKU...5{"data":{"col":9,"row":7,"turn":"B"},"type":"update"}
C: GMKU...){"data":{"col":4,"row":2},"type":"place"}
S: GMKU...5{"data":{"col":4,"row":2,"turn":"W"},"type":"update"}
(对方落子)
S: GMKU....{"data":"B","type":"win"}
```

本轮游戏结束。

这个时候，另外一个客户端发出了开新局的请求，服务器开局。

```
(对方发出开局请求)
S: GMKU....{"type":"hello"}
S: GMKU....{"data":"B","type":"color"}
S: GMKU....{"data":{"board":[[" "," "," "," "," "," "," "," "," "," "," "," "
","," "," "," "],[" "," "," "," "," "," "," "," "," "," "," "," "," "," "," "],
[" "," "," "," "," "," "," "," "," "," "," "," "," "," "," "],[" "," "," ","
","," "," "," "," "," "," "," "," "," "," "," "],[" "," "," "," "," "," "," ","
","," "," "," "," "," "," "," "],[" "," "," "," "," "," "," "," "," "," "," ","
","," "," "," "],[" "," "," "," "," "," "," "," "," "," "," "," "," "," "," "],
[" "," "," "," "," "," "," "," "," "," "," "," "," "," "," "],[" "," "," ","
","," "," "," "," "," "," "," "," "," "," "," "],[" "," "," "," "," "," "," ","
","," "," "," "," "," "," "," "],[" "," "," "," "," "," "," "," "," "," "," ","
","," "," "," "],[" "," "," "," "," "," "," "," "," "," "," "," "," "," "," "],
[" "," "," "," "," "," "," "," "," "," "," "," "," "," "," "],[" "," "," ","
","," "," "," "," "," "," "," "," "," "," "," "],[" "," "," "," "," "," "," ","
","," "," "," "," "," "," "
"]],"col":-1,"row":-1,"turn":"B"},"type":"update"}
```

接着，就是新的一轮游戏。

```
C: GMKU...){"data":{"col":7,"row":7},"type":"place"}
S: GMKU...5{"data":{"col":7,"row":7,"turn":"W"},"type":"update"}
(对方落子)
S: GMKU...5{"data":{"col":6,"row":6,"turn":"B"},"type":"update"}
...(轮流落子)...
C: GMKU...*{"data":{"col":10,"row":4},"type":"place"}
S: GMKU...6{"data":{"col":10,"row":4,"turn":"W"},"type":"update"}
S: GMKU....{"data":"B","type":"win"}
```

本轮游戏结束。

这个时候，客户端发出了开新局的请求。

```
C: GMKU....{"type":"new"}
```

服务器开局，然后就是新的一轮游戏。

```
S: GMKU....{"type":"hello"}
...(又是一局)...
S: GMKU....{"data":"W","type":"win"}
```

客户端断开了连接，游戏结束。

算法

本项目涉及到的算法包括五子棋获胜或平局检测以及危险检测，由`Engine.cpp`实现，客户端模块和服务端模块共用它。

获胜或平局检测

整体检测的算法十分简单，对于每一个点，计算它向右、下、右下、左下方向是否形成五个连续的相同颜色的子。如果存在，那么该颜色的玩家获胜，如果棋盘已满，那么平局。

还有一种增量检测的办法，只需要检测最后一个子的米字型是否形成五个连续的相同颜色的子。

危险检测

危险检测算法和检测获胜或平局的算法相似，对于每一个可以落子的点，检测假设对方在这个位置落子，该位置是否会出现至少两个无阻挡的三个连续子，至少一个无阻挡的三个连续子以及一个有单侧阻挡的四个连续子，至少一个无阻挡的四个连续子，获胜等情况。