

C++实训报告

- **设计题目：植此青绿**

姓名：刘胜杰 学号：202330551041

班级：软件工程 1 班 指导教师：杜卿

- **设计报告**

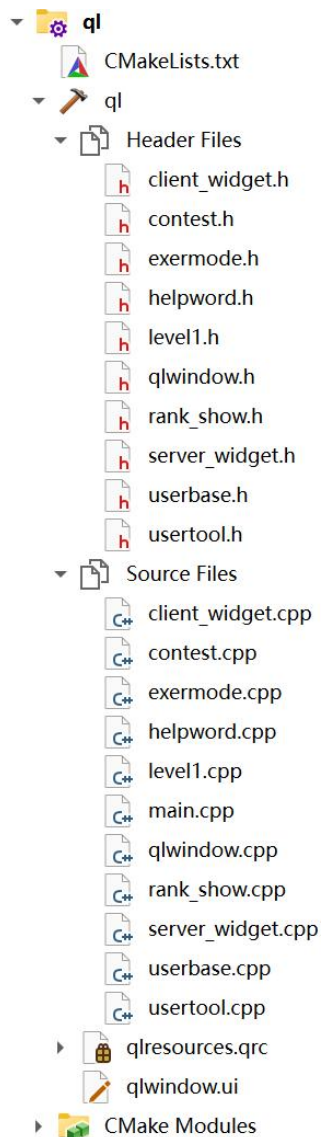
1.作品描述

本作品为基于最强大脑第十一季第三期的比赛项目“植此青绿”进行复刻并扩展的项目。主要玩法为将题目所给出的若干不同大小的矩阵全部放置到地图的对应位置上，使得地图的每一个格子上的树木数量正确。本项目在实现了原本的能够和地图进行交互、提交答案并上传用户数据、观看用户排行榜的功能的基础上，增添了随机生成地图、用户排行榜多元化、关卡提示、生成题目时写入题库并从题库随机选取题目、基于 `socket` 的同一局域网下的单一客户端与服务端进行连接并聊天的聊天室等功能。本作品完全基于 Qt6.5.3 进行开发。

2.系统设计说明

本项目主要分为四个大模块，分别是练习模式、比赛模式、排行榜、基于 `socket` 的联网聊天室。

对于项目架构：



一共定义了十个类，其中，**qlwindow** 为主页面，**exermode** 为练习模式选择关卡页面，**level1** 为练习模式游戏页面，**rank_show** 为排行榜页面，**sever_widget** 和 **client_widget** 为聊天室的服务端与客户端，**helpword** 为帮助说明，**userbase** 为用户基类，用于用户字段创建，**usertool** 为游戏交互逻辑会用到的类，**contest** 为比赛模式游戏页面。

在进入主页面后可选择进入练习模式、比赛模式、排行榜、帮助文档、打开聊天室的服务端与客户端。

对于游戏交互的核心逻辑，本项目没有使用拖拽，而是采用了通过鼠标点击确定需要放置的矩阵的左上角坐标，然后在点击需要放置的矩阵的按键来实现区间内种树的功能。而对于提交答案判断正误和用户数据的建立逻辑，在这里的设计是在当前关卡提交并判断正确后，跳出 **QMessageBox** 供玩家写入用户昵称，根据本关数据创建用户字段。

- 2.1 练习模式

练习模式主要由五个基础关卡和一个随机从题库选取题目的关卡组成，游戏关卡的核心生成逻辑均为读取存储地图的 **txt** 文件，前五个基础关卡的地图均为加入 **qrc** 的 **txt** 文件，最后一个关卡随机从题库进行读取的关卡从专门的排行榜文件中进行读取文件并随机选取。

练习模式是整个游戏模式的基础框架，主要模块包括：关卡矩阵放置功能、撤回功能、重置功能、计时器模块、提交答案并判断正误、创建用户字段并将用户数据写入排行榜。

关于游戏页面的 UI 设计，在这里使用了 Qt 封装好的控件类：**QLabel**、**QPushButton**、**Qpalette**、**QMessageBox**、**QStack**、**Qvector**、**QPixmap**、**QVariant**。此外，为了方便实现撤销功能和创建用户字段，使用了自定义类：**userbase**、**cancelbox**。

下面是关键变量：

```
QPushButton*buttonback=NULL;//返回

QVector<QVector<QLabel*>>matrix1;//存储格子的图片
QVector<QVector<QLabel*>>matrix2;//存储格子的原本数据和用户放置数据
QVector<userbase>users;//用户字段

QVector<QLabel*>choices;//显示每种矩阵还剩多少

QLabel*selectedlabel=NULL;//选中的起点
int selectedlabel_x=0,selectedlabel_y=0;

QChar char_;//读文件字符
int size=6;

QTimer timer;
QLabel*label_time=NULL;
int time_total=0;

QVBoxLayout*vlayout1=NULL;
QVBoxLayout*vlayout2=NULL;
QGridLayout*glayout=NULL;
QHBoxLayout*hlayout1=NULL;
QHBoxLayout*hlayout2=NULL;
QHBoxLayout*hlayout3=NULL;
QHBoxLayout*hlayout_total=NULL;//总布局

QPushButton*button1=NULL;//选1*1
QPushButton*button2=NULL;
QPushButton*button3=NULL;
QPushButton*button4=NULL;
QPushButton*button_cancel=NULL;//选撤回
QPushButton*button_submit=NULL;//提交
QPushButton*button_reset=NULL;//重置地图

QStack<cancelbox>cancelboxer;//存储操作方便撤回
```

页面逻辑的具体实现在设计实现与分析中展开。

– 2.2 比赛模式

比赛模式在套用练习模式的界面 UI 的基础上（QWidget 类：contest），除去了读取地图文件进行游戏地图加载的逻辑，设计了一套随机生成算法进行地图的生成，并将随机生成的地图写入题库文件夹。此外，考虑到比赛模式可能过于困难，不易做出，在比赛模式中加入了关卡提示功能，可提示玩家部分矩阵的正确放置位置，降低游玩难度。

比赛模式的关键变量在练习模式的基础上增添了供随机生成算法使用的部分：

```

QVector<QVector<int>>>valuestorage;//存储生成的地图
QVector<rangedot>dot_legal_range;//存储每种矩阵可放置的合法区间
QVector<int>candimap={4,3,3,2,2,1,1};//矩阵边长大小
bool have_four=false;//生成时是否已经出现4
int sum=0;

QVariant sx4,sy4;//已经出现4的左上角坐标

```

随机算法的实现在设计实现与分析中展开。

- 2.3 排行榜

在原有的排行榜基础上，本项目的排行榜实现了支持练习模式、比赛模式的两种模式，并且同时在单一模式下可按照 id 昵称、通关次数、最短通关时间三种排序方式进行排行榜显示的多元化排行榜功能。整体排行榜采用 QTableView 进行设置，以 QStandardItemModel 进行用户数据的存放。

关键变量：

```

QPushButton*button_back=NULL;//返回
QPushButton*button_common=NULL;//练习模式
QPushButton*button_boss=NULL;//比赛模式
QPushButton*button_id=NULL;//按id排序
QPushButton*button_pass_time=NULL;//按通关次数排序
QPushButton*button_timeuse=NULL;//按最短用时牌序

QStandardItemModel*usermodel=NULL;//存放用户数据条目，由tableview管理
QTableView*tableview=NULL;//排行榜显示设置

QVBoxLayout*vlayout1=NULL;
QHBoxLayout*hlayout1=NULL;
QHBoxLayout*hlayout2=NULL;
QHBoxLayout*hlayout3=NULL;

QVector<userbase>users;//从排行榜文件读取用户数据并存储

```

- 2.4 聊天室

该聊天室基于 QTcpSocket 建立,服务端和客户端各占用一个页面，在同一局域网下，服务端开始监听后，在客户端页面输入服务端的 IP 地址并请求连接。完成链接后客户端和服务端即可在聊天页面进行对话。

关键变量：

客户端:

```
QLineEdit *server_address=NULL;//要链接的服务器地址
QLineEdit *server_port=NULL;//对应服务器端口

QPushButton *button_connect=NULL;//请求连接
QPushButton *button_disconnect=NULL;//断开连接

QTextEdit *communicationLog=NULL;//聊天框
QLineEdit *message_line=NULL;//聊天信息输入栏
QPushButton *button_send=NULL;//发送按钮

QLabel *connectionStatusLabel=NULL;//显示连接状态
QTcpSocket *socket=NULL;//关键通信媒介
```

服务端:

```
QLineEdit *port=NULL;//端口号
QLineEdit *host_address=NULL;//主机地址

QPushButton *button_start=NULL;//开始监听
QPushButton *button_stop=NULL;//停止监听

QTextEdit *communicationLog=NULL;//聊天框
QLineEdit *message_line=NULL;//信息输入栏
QPushButton *button_send=NULL;//发送按钮

QLabel *serverStatusLabel=NULL;//连接状态
QTcpServer *server=NULL;//服务器
QList<QTcpSocket*> clients;//与主机建立了连接的客户端
```

3.实现环境

本项目完全基于 Qt6.5.3 进行开发，开发平台为 Qt Creator，Kit: Desktop Qt6.5.3 MinGW 64-bit，项目使用 CMake 进行部署。运行版本需使用 Qt 自带链接终端运行 windeployqt 命令将环境依赖装进补全。


```
Creating qt_ar.qm...
Creating qt_bg.qm...
Creating qt_ca.qm...
Creating qt_cs.qm...
Creating qt_da.qm...
Creating qt_de.qm...
Creating qt_en.qm...
Creating qt_es.qm...
Creating qt_fi.qm...
Creating qt_fr.qm...
Creating qt_gd.qm...
Creating qt_he.qm...
Creating qt_hu.qm...
Creating qt_it.qm...
Creating qt_ja.qm...
Creating qt_ko.qm...
Creating qt_lv.qm...
Creating qt_pl.qm...
Creating qt_ru.qm...
Creating qt_sk.qm...
Creating qt_uk.qm...
Creating qt_zh_TW.qm...
```

补全环境依赖后 exe 文件即可运行，打包发送。

4.设计实现及分析

首先是基本页面的切换和背景、大小设置，在本项目中，所有可通过 QpushButton 进行切换的页面都是上一个页面的子成员，在父页面点击后，父页面调用 close 函数关闭，子页面调用 show 函数显示，在子页面点击返回后，子页面释放信号 back（），父页面接收到信号触发槽函数关闭子页面，显示主页面。

例如主页面前往练习模式：

```
void qlwindow::to_exermode()//前往练习模式
{
    this->close();
    page1->show();
}

connect(button1,&QPushButton::clicked,this,&qlwindow::to_exermode)
connect(page1,SIGNAL(back()),this,SLOT(fromexer_returntothis()));
,
```

```

void qlwindow::fromexer_returntothis()//练习模式返回主页面
{
    page1->close();
    this->show();
}

void exermode::returntomain()
{
    emit this->back();
}

```

对于每个页面的背景和大小设置，以及 QWidget 的窗口标题和窗口图标：
在构造函数中，窗口大小使用 `setFixedSize` 进行大小固定，使其不会随鼠标拖动而改变，窗口标题用 `setIcon` 传入图片路径，背景使用 `QPalette` 调色板进行设置，以练习模式为例：

```

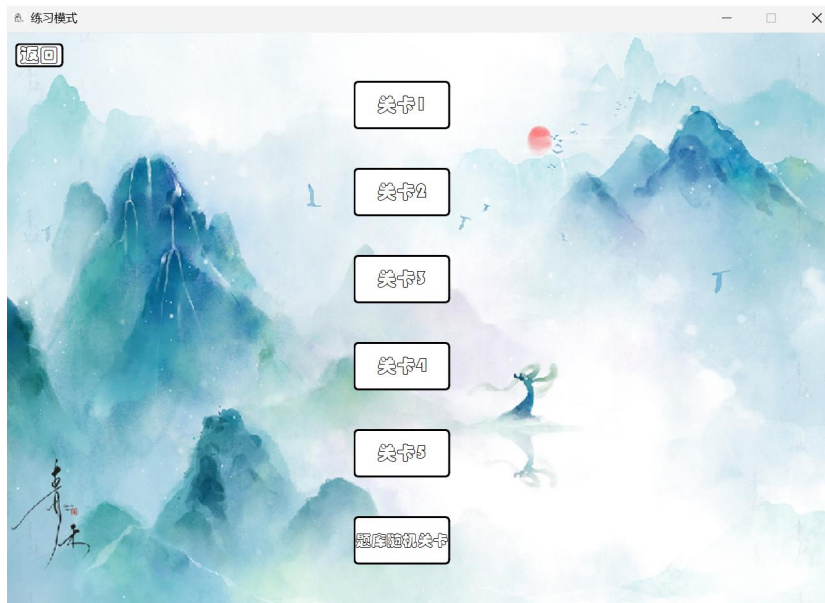
setFixedSize(863,600);//固定大小使得页面不会随鼠标拖动而改变大小

setWindowIcon(QIcon(":/images/resources/images/common.png")); //窗口图标
setWindowTitle(QString("练习模式")); //窗口标题

QImage background(":/images/resources/images/background1.jpg");
background=background.scaled(this->width(),this->height(),Qt::IgnoreAspectRatio); //图片适应窗口大小
QPalette palette;
palette.setBrush(this->backgroundRole(),QBrush(background));
this->setPalette(palette); //使用调色板进行背景设置

```

呈现的效果如下：



– 4.1 练习模式

1.对于矩阵在地图上的放置功能，这里采用的是鼠标点击需放置矩阵的左上角坐标来进行放置位置确认，然后选择矩阵进行放置，这也是游戏交互的核心逻辑，首先定义鼠标事件，在点击后设置 `selectedlabel` 并存储坐标：

```
void level1::mousePressEvent(QMouseEvent*event)
{
    if(event->button()==Qt::LeftButton)
    {
        for(int i=0;i<size;i++)
        {
            for(int j=0;j<size;j++)
            {
                if(matrix1[i][j]->geometry().contains(event->pos()))
                {
                    selectedlabel=matrix2[i][j];
                    selectedlabel_x=i;selectedlabel_y=j;
                    return;
                }
            }
        }
    }
}
```

然后是点击选择需要放置的矩阵，以 4×4 的矩阵为例，在点击后触发槽函数，先判断 `selectedlabel` 是否为空，不为空则判断起点坐标加上矩阵大小是否会越界，不会越界则以 `selectedlabel` 为起点进行遍历区间加一，同时设置 `selectedlabel` 为空，并将该操作存储进 `cancelboxer`，矩阵剩余数量减一，若 `selectedlabel` 为空或者会越界，则跳出 `QMessageBox` 提示：

```

if(selectedlabel!=NULL)//先判断是否selectedlabel为空
{
    QVariant value1=choices[3]->property("rest_num");
    int cur=value1.toInt();
    if(selectedlabel_x+3<size&&selectedlabel_y+3<size&&cur!=0)//判断是否越界
    {
        for(int i=selectedlabel_x;i<=selectedlabel_x+3;i++)
        {
            for(int j=selectedlabel_y;j<=selectedlabel_y+3;j++)//区间加一
            {
                QVariant curvalue=matrix2[i][j]->property("now_num");
                int data=curvalue.toInt();
                data+=1;
                matrix2[i][j]->setProperty("now_num",QVariant(data));
                QVariant storedvalue=matrix2[i][j]->property("now_num");
                matrix2[i][j]->setText(storedvalue.toString());
            }
        }
        cancelbox box(selectedlabel,selectedlabel_x,selectedlabel_y,4);//存储操作
        cancelboxer.push((box));
        selectedlabel=NULL;

        //该矩阵剩余数量减一
        QVariant value=choices[3]->property("rest_num");
        int now=value.toInt();
        now-=1;
        choices[3]->setProperty("rest_num",QVariant(now));
        choices[3]->setText(QString::number(now));
    }
    else//越界则提示
    {
        QMessageBox messageBox;
        messageBox.setIcon(QMessageBox::Information);
        messageBox.setWindowTitle("提示");
        messageBox.setWindowIcon(QIcon(":/images/resources/images/common.png"));
        messageBox.setText("无效操作");
        messageBox.setStandardButtons(QMessageBox::Ok);
        messageBox.setDefaultButton(QMessageBox::Ok);

        messageBox.exec();
    }
}
else//为空则提示无效
{
    QMessageBox messageBox;
    messageBox.setIcon(QMessageBox::Information);
    messageBox.setWindowTitle("提示");
    messageBox.setWindowIcon(QIcon(":/images/resources/images/common.png"));
    messageBox.setText("未选定起点");
    messageBox.setStandardButtons(QMessageBox::Ok);
}

```

2.对于撤回功能，在点击 pushbutton 触发槽函数后，先判断存储操作的 QStack 是否为空，为空则跳 QMessageBox 提示，不为空则上一次操作出栈，进行区间减一同时矩阵数量加一：

```

if(cancelboxer.empty())//操作为空
{
    QMessageBox messageBox;
    messageBox.setIcon(QMessageBox::Information);
    messageBox.setWindowTitle("提示");
    messageBox.setWindowIcon(QIcon(":/images/resources/images/common.png"));
    messageBox.setText("无效操作");
    messageBox.setStandardButtons(QMessageBox::Ok);
    messageBox.setDefaultButton(QMessageBox::Ok);

    messageBox.exec();
}
else//操作不为空
{
    cancelbox box=cancelboxer.top();//取出上一次操作
    cancelboxer.pop();
    for(int i=box.selectedlabel_x;i<=box.selectedlabel_x+box.size-1;i++)
    {
        for(int j=box.selectedlabel_y;j<=box.selectedlabel_y+box.size-1;j++)
        {
            QVariant curvalue=matrix2[i][j]->property("now_num");
            int data=curvalue.toInt();
            data-=1;
            matrix2[i][j]->setProperty("now_num",QVariant(data));
            QVariant nowvalue=matrix2[i][j]->property("now_num");
            matrix2[i][j]->setText(nowvalue.toString());
        }
    }
    //进行区间减一
    QVariant value=choices[box.size-1]->property("rest_num");
    int now=value.toInt();
    now+=1;//矩阵数量加一
    choices[box.size-1]->setProperty("rest_num",QVariant(now));
    choices[box.size-1]->setText(QString::number(now));
}

```

3.对于重置功能，遍历全图将 label 存储的值初始化为 0，同时 cancelbox 清空，矩阵数量初始化。

4.对于计时器，使用 QTimer 设置，当进入游戏子页面的时候，调用子页面的 timer 的 start 函数，并设置参数为 1000ms，表示每 1000ms 触发一次 timeout 信号，当退出页面的时候停止计时，调用 stop 函数，当需要获取计时数据的时候，使用 timer.interval 来访问。在页面的显示计时上，先设置 connect 链接 timer 的 timeout 信号和显示时间的槽函数。当每秒触发槽函数时进行时间设置：

```

void level1::show_time()
{
    time_total+=timer.interval();//获取当前时间
    label_time->setText(QString::number(time_total/1000.0,'f',2)+'s');//设置为xx.00s的样式
}

```

5.对于提交答案并判断正误，以及创建用户字段写入数据，点击提交 pushbutotn 触发槽函数后，先遍历全图进行答案判断，判断标准是用户放置完的每一个格子的数据是否和格子原本存储的数据一致，如

果不一致则跳出 QMessageBox 提示，如果一致则清空 cancelbox，停止计时器，跳出写入用户 id 的提示，在写入 id 后点击 ok 后，从 Json 格式的排行榜文件中读取数据，先使用 QByteArray 接受文件 read 读取的数据，然后转化为 QJsonDocument 变量 docread，先判断是否为空，为空即代表文件为空，则创建一个空的 QJsonArray，写入新用户数据，然后转化为 QJsonDocument，再往文件中 write 进去；如果 docread 不为空并且无重名，则将 docread 转化为 QJsonArray，然后往里 append 用户数据，然后转化为 QJsonDocument 写入文件；如果 docread 不为空但有重名，则更新最优数据，创建新的 QJsonArray，写入所有用户数据再写入 QJsonArray，转化为 QJsonDocument 再写入文件：

```
cancelboxer.clear();
bool flag=true;
for(int i=0;i<size;i++)//遍历判断正误
{
    for(int j=0;j<size;j++)
    {
        QVariant curvalue=matrix2[i][j]->property("now_num");
        int data1=curvalue.toInt();
        QVariant storedvalue=matrix2[i][j]->property("storedvalue");
        int data2=storedvalue.toInt();
        if(data1!=data2)
        {
            flag=false;
            break;
        }
    }
}

if(timer.isActive())
{
    timer.stop();
}

QMessageBox messageBox;
messageBox.setIcon(QMessageBox::Information);
messageBox.setWindowTitle("提示");
messageBox.setWindowIcon(QIcon(":/images/resources/images/common.png"));
messageBox.setStandardButtons(QMessageBox::Ok);
messageBox.setDefaultButton(QMessageBox::Ok);
if(flag)//正确则创建用户字段写入文件
{
    QString timetotal=QString::number(time_total/1000.0);
    messageBox.setInformativeText("你成功了! 用时: "+timetotal+".00s");
    bool ok;
    QString id = QDialog::getText(nullptr,
                                    "用户登录",
                                    "请输入用户ID: ",
                                    QLineEdit::Normal,
                                    "",
                                    &ok);
}
```

```

QFile fileread("./rank/common.json");
if (!fileread.open(QIODevice::ReadOnly)) {
    qDebug() << "Failed to open file for read: 111" ;
}

QByteArray jsonData = fileread.readAll();
fileread.close();

QJsonDocument docread = QJsonDocument::fromJson(jsonData);

userbase usernow(id,time_total/1000,1);

//表示文件为空，未写过数据
if (docread.isNull()) {
    QJsonArray usersArray=QJsonArray();
    usersArray.append(usernow.toJsonObject());
    QJsonDocument docwrite(usersArray);

    QFile filewrite("./rank/common.json");
    if (!filewrite.open(QIODevice::WriteOnly|QIODevice::Truncate )) {
        qDebug() << "Failed to open file for writing: 111" ;
    }
    filewrite.write(docwrite.toJson());
    filewrite.close();
}

//文件不为空，写过数据
else //if(!docread.isNull())
{
    //先判断是否有重名
    users=userbase::loadUsersFromJson("./rank/common.json");
    bool check=true;
    for(userbase& user:users)
    {
        if(user.get_id()==id)
        {
            user.change_time(std::min(user.get_time(),usernow.get_time()));
            user.add_pass_time();
            check=false;
            break;
        }
    }
}

```



```

//找到重名的，直接改变原数据后再写入
if(!check)
{
    QJsonArray userarray=QJsonArray();
    for(const userbase&user:users)
    {
        userarray.append(user.toJsonObject());
    }
    QJsonDocument doc(userarray);

    QFile filewrite("./rank/common.json");
    if (!filewrite.open(QIODevice::WriteOnly|QIODevice::Truncate )) {
        qDebug() << "Failed to open file for writing:222";
    }

    filewrite.write(doc.toJson());
    filewrite.close();
}

//未找到重名的，在原数据后面写入新数据
else //if(check)
{
    QJsonArray userarray=docread.array();
    userarray.append(usernow.toJsonObject());
    QJsonDocument doc(userarray);

    QFile filewrite("./rank/common.json");
    if (!filewrite.open(QIODevice::WriteOnly|QIODevice::Truncate )) {
        qDebug() << "Failed to open file for writing:333";
    }

    filewrite.write(doc.toJson());
    filewrite.close();
}

```

此外，由于练习模式中设置了六个关卡，五个基础关卡和一个随机题库关卡，为了提高代码复用性，减少冗余代码，练习模式的所有游戏页面均为 level1，在点击选择关卡的时候，触发槽函数切换页面会访问不同文件，根据文件地图数据初始化游戏页面：


```

void level1::loadfile(const QString filePath)//打开地图文件初始化
{
    QFile file(filePath);
    if(!file.open(QIODevice::ReadOnly | QIODevice::Text))
    {
        qCritical() << "Failed to open file: " << file.errorString();
        exit(1);
    }
    QTextStream in(&file);

    for(int i=0;i<size;i++)
    {
        for(int j=0;j<size;j++)//读取字符并根据地图数字来给格子贴图
        {
            while(!in.atEnd())
            {
                in>>char_;
                if(char_.isDigit())break;
            }
            if(char_.isDigit())
            {
                int data=char_.digitValue();
                if(data==0)matrix1[i][j]->setPixmap(QPixmap(":/images/resources/images/tree0.jpg"));
                else if(data==1||data==3)matrix1[i][j]->setPixmap(QPixmap(":/images/resources/images/tree13.jpg"));
                else if(data==2)matrix1[i][j]->setPixmap(QPixmap(":/images/resources/images/tree2.jpg"));
                else if(data==4)matrix1[i][j]->setPixmap(QPixmap(":/images/resources/images/tree4.jpg"));
                matrix1[i][j]->setScaledContents(true);

                int now=0;
                matrix2[i][j]->setProperty("now_num",QVariant(now));
                matrix2[i][j]->setProperty("storedvalue",QVariant(data));
                QVariant stored=matrix2[i][j]->property("now_num");
                matrix2[i][j]->setText(stored.toString());
            }
        }
    }
    file.close();
}

```

- 4.2 比赛模式

比赛模式相较于练习模式只做了两点改动，一个是重置按钮被设置成了随机生成新地图并写入图库文件夹，一个是增加了提示功能。

1.对于随机生成算法，核心思想为在当前需要放置的矩阵放置之前，对地图进行遍历，枚举所有当前矩阵可放置的合法区域（即防止之后不会出现不合法的情况，如出现两个四或者出现大于四的情况），然后再所有合法区域内随机选取进行放置。这样子虽然单次生成地图的复杂度较高，但是因为不合法而重新生成的次数将大大减少，生成地图更为稳定。

1.1 首先设置地图的检查函数，判断地图合法的判断标准是，全图数字小于等于四，并且有且仅有一个四，且全局数字和为 44：

```

bool contest::check_fit_map()//地图合法条件：有且仅有一个四，所有数字不得大于四，地图数字和为44
{
    bool flag=true;int num_4=0;
    for(int i=0;i<size&&flag;i++)
    {
        for(int j=0;j<size;j++)
        {
            sum+=valuestorage[i][j];
            if(valuestorage[i][j]==4)num_4++;
            if(valuestorage[i][j]>4||num_4>1)
            {
                flag=false;
                break;
            }
        }
    }
    if(flag&&num_4==1&&sum==44)return true;
    else
    {
        sum=0;
        qDebug() << "fail to generate";
        return false;
    }
}

```

1.2 对区间进行+1 的操作函数，在这里由于后续的地图生成逻辑需要分为已生成四和未生成四的情况，因此需要在出现四的时候进行布尔值更改：

```

void contest::adddot(int sx,int sy,int width,int height)//区间加一
{
    for(int i=sx;i<sx+width;i++)
    {
        for(int j=sy;j<sy+height;j++)
        {
            valuestorage[i][j]++;
            if(valuestorage[i][j]==4)
                have_four=true;
        }
    }
    qDebug() << "add dot over";
}

```

1.3 对区间进行检查是否合法的函数，在这里需要进行分类，由于需要保证矩阵放置后依旧是合法的情况，因此在没有出现四的时候，区间合法的判定标准为区间内最多只有一个三，已经出现四了之后，区间合法的判定标准为区间内没有三，这样才能保证放置了矩阵之后地图依旧合法：

```

bool contest::check_if_more_3(int sx,int sy,int width,int height)//没有四的时候区间内至多一个三
{
    int num_3=0;
    for(int i=sx;i<sx+width;i++)
    {
        for(int j=sy;j<sy+height;j++)
        {
            if(valuestorage[i][j]==3)num_3++;
        }
    }
    if(num_3>1)
    {
        qDebug() << "check1false";
        return true;
    }
    else
    {
        qDebug() << "check1ok";
        return false;
    }
}

bool contest::check_if_3(int sx,int sy,int width,int height)//有四的时候区间内不能有三
{
    bool flag=true;
    for(int i=sx;i<sx+width&&flag;i++)
    {
        for(int j=sy;j<sy+height;j++)
        {
            if(valuestorage[i][j]==3)
            {
                flag=false;
                break;
            }
        }
    }
    if(flag)
    {
        qDebug() << "check2ok";
        return true;
    }
    else
    {
        qDebug() << "check2false";
        return false;
    }
}
}

```

1.4 在已给范围内遍历进行合法区间的寻找，首先需要特判区域是否能容纳当前矩阵的大小，不能就返回，然后根据将矩阵的大小的是否已经出现四进行区域遍历，寻找合法区间，然后将所有合法区间放进vector内，特别的，虽然这个区间判定可能出现将四的单位划定为一的合法区间的情况，但是考虑到 1/36 的概率，并且即便不合法也可重新生成，在这里不单独对一进行处理：

```

void contest::add_legal_range(int sx,int sy,int fx,int fy,int width,int height)//在一块区域内寻找对width*height的矩阵合适的所有区间
{
    if(sx+width-1>fx||sy+height-1>fy)//判断是否越界
    {
        qDebug() << "add range false";
        return;
    }
    for(int i=sx;i+width-1<=fx;i++)
    {
        for(int j=sy;j+height-1<=fy;j++)
        {
            if((!have_four)?(!check_if_more_3(i,j,width,height)):(!check_if_3(i,j,width,height)))//分情况进行区间检查
            {
                dot_legal_range.push_back(rangedot(i,j,i+width-1,j+height-1));
            }
        }
    }
}
}

```

1.5 区间寻找范围的划定，对于还没出现四的时候，寻找范围即为全局，因为全局都可能出现四；对于已经出现四的情况，寻找范围为全局除开四的单位以外组成的“回”字的四条边，这样能保证不会出现数字大于四的情况，特别地，当四处于地图边界的时候，只有一个或两个区间或三个范围是有效的，不影响寻找范围的划定，因为不合法的范围在上一个函数开始部分 return 了：

```

void contest::find_legal_range(int width,int height)
{
    //没有四的时候对全局进行搜索
    if(!have_four)
    {
        add_legal_range(0,0,size-1,size-1,width,height);
        qDebug() << "find1ok";
    }
    else
    {
        //已经有4的时候，先找到4的坐标
        int x4,y4;bool is4=true;
        for(int i=0;i<size&&is4;i++)
        {
            for(int j=0;j<size;j++)
            {
                if(valuestorage[i][j]==4)
                {
                    x4=i;y4=j;
                    is4=false;
                    break;
                }
            }
        }
        //搜索包围着4的“回”字的四条边
        add_legal_range(0,0,std::min(x4-1,0),size-1,width,height);
        add_legal_range(0,0,size-1,std::min(y4-1,0),width,height);
        add_legal_range(std::max(x4+1,size-1),0,size-1,size-1,width,height);
        add_legal_range(0,std::max(y4+1,size-1),size-1,size-1,width,height);
        qDebug() << "find2ok";
    }
}
}

```

1.6 地图生成函数，开头设置随机数种子，在生成时逐个取出需放置矩阵，并进行合法区间的寻找，如果没有合法区间，即当前地图生成

已失效，则跳出循环重新开始。然后用 rand 函数进行随机选取并放置，每次操作完一个矩阵将存储当前矩阵的合法区间的 vector 给 clear。生成完成后根据地图的单位数字进行图片粘贴，在比赛模式里，将 0 和 2 的图标统一为○：

```
void contest::generatingmap()
{
    srand(QTime(0,0,0).secsTo(QTime::currentTime())); //随机数种子
    valuestorage.resize(size,QVector<int>(size,0)); //先初始化为0防止访问非法内存
    qDebug() << "resize over start generating";
    for(int k=0;k<1e4;k++) //足够多的循环次数保证找出解
    {
        for(int i=0;i<candimap.length();i++) //逐个取出矩阵，进行合法区间寻找和放置
        {
            int border=candimap[i];
            int border_width=std::min(border,size);
            int border_height=std::min(border,size);

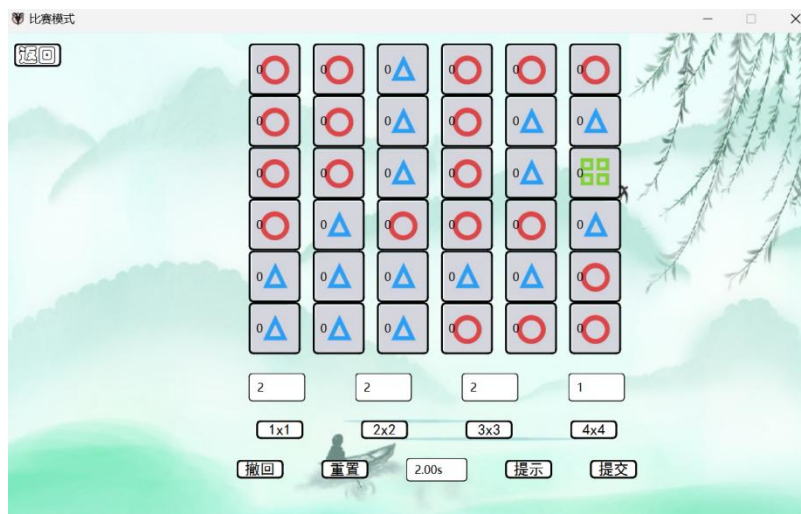
            find_legal_range(border_width,border_height);
            if(dot_legal_range.length()==0)
            {
                break;
            }
            int selectmap_num=std::rand()%dot_legal_range.length();
            //从该矩阵的所有合法区间进行随机选取然后放置

            adddot(dot_legal_range[selectmap_num].sx,dot_legal_range[selectmap_num].sy,border_width,border_height);
            if(border_height==4&&border_width==4) //单独为4*4矩阵设置提示功能
            {
                sx4=dot_legal_range[selectmap_num].sx+1;
                sy4=dot_legal_range[selectmap_num].sy+1;
            }
            dot_legal_range.clear();
        }
        if(check_fit_map()) //本次地图合法就结束循环
        {
            qDebug() << "OK";
            break;
        }
        else //本次地图不合法就重新生成
        {
            valuestorage.clear();
            valuestorage.resize(size,QVector<int>(size,0));
            have_four=false;
            qDebug() << "failed";
        }
    }
}

for(int i=0;i<size;i++) //同练习模式，但是图标有变化
{
    for(int j=0;j<size;j++)
    {
        if(valuestorage[i][j]==0||valuestorage[i][j]==2) matrix1[i][j]->setPixmap(QPixmap(":/images/resources/images/tree2.jpg"));
        else if(valuestorage[i][j]==1||valuestorage[i][j]==3) matrix1[i][j]->setPixmap(QPixmap(":/images/resources/images/tree13.jpg"));
        else matrix1[i][j]->setPixmap(QPixmap(":/images/resources/images/tree4.jpg"));
        matrix1[i][j]->setScaledContents(true);

        int now=0;
        matrix2[i][j]->setProperty("now_num",QVariant(now));
        matrix2[i][j]->setProperty("storedvalue",QVariant(valuestorage[i][j]));
        QVariant stored=matrix2[i][j]->property("now_num");
        matrix2[i][j]->setText(stored.toString());
    }
}
```

地图生成效果如下：



经验证地图合法有解。

2.写入题库文件夹

在随机生成算法生成新地图后，调用生成新地图文件的函数来写入地图文件：


```

void contest::writefile()//将随机地图写入文件
{
    QString folderPath = "./randommap";//打开文件夹
    QDir directory(folderPath);
    if (!directory.exists() ) {
        qDebug() << "Folder not found";
        return;
    }
    directory.setNameFilters(QStringList());//获取目录下所有文件
    QFileInfoList fileList = directory.entryInfoList();
    int filenum=fileList.count();
    //根据已有文件数量来微信文件命名
    QString fileName = "maze%1.txt";
    QString filename=fileName.arg(++filenum);
    QString filePath = folderPath + "/" + filename;//新文件路径

    QFile file(filePath);
    if (file.exists()) {
        file.remove();
    }
    if (!file.open(QIODevice::WriteOnly | QIODevice::Text)) {
        qDebug() << "Failed to open the file for writing: " << file.errorString();
        return;
    }
    QTextStream out(&file);
    //逐个数据写入文件
    for(int i=0;i<size;i++)
    {
        for(int j=0;j<size;j++)
        {
            out<<valuestorage[i][j];
            out<<' ';
        }
        out<<'\n';
    }
    file.flush();
    file.close();
}

```

然后是在练习模式设置题库题目的抽取并加载文件，首先获取文件夹路径，然后获取文件夹路径下的文件列表，并基于文件数量进行随机选取，然后调用游戏页面的 loadfile 函数进行地图加载：

```

void exermode::to_random()
{
    QString folderPath = "./randommap";//随机题库文件夹
    QDir directory(folderPath);

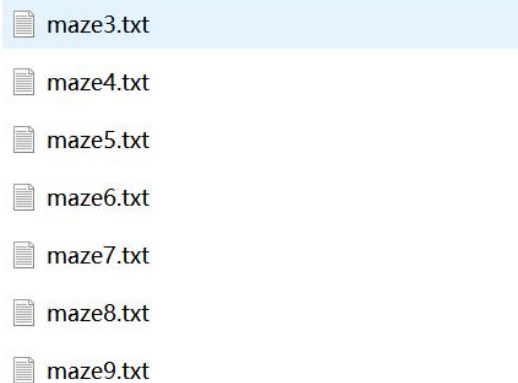
    if (!directory.exists() || !directory.isReadable()) {
        qDebug() << "Folder not found or not readable.";
        return;
    }

    //获取目录下所有文件
    QStringList fileNames = directory.entryList(QDir::Files | QDir::NoDotAndDotDot);
    if (fileNames.isEmpty()) {
        qDebug() << "No files found in the folder.";
        return;
    }
    //根据文件数量随机生成读取的文件number
    int randomIndex = QRandomGenerator::global()->generate() % fileNames.size();
    QString filename = fileNames[randomIndex];
    QString filepath=directory.filePath(filename);//读取文件

    this->page1->loadfile(filepath);
    this->close();
    this->page1->show();
}

```

题库文件夹效果如下：



3.提示功能设置

由于在比赛模式中，难度可能过高，但考虑到每一步都进行提示不符合游戏逻辑，并且倘若玩家策略错误会导致实时提示逻辑过于复杂，因此只制作了对于当前地图的 4*4 的地图的位置提示，即提示 4 * 4 地图的左上角坐标。

基本逻辑为，在上一周设计的随机生成算法里，当地图为 4 * 4 时，记录下所选取的随机合法区间的左上角坐标，并使用 QMessageBox 提示：

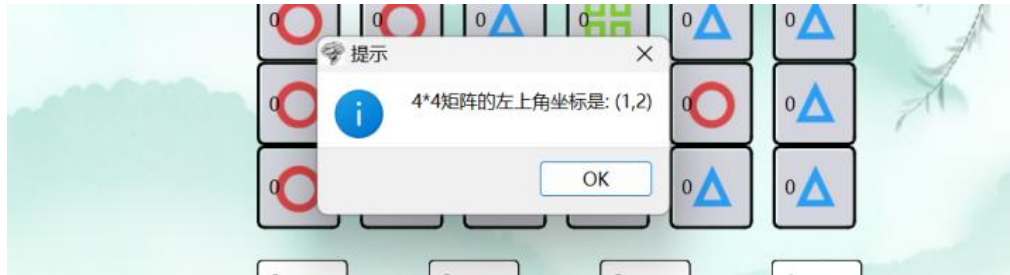
```

void contest::choose_tips()//提示4*4矩阵左上角坐标
{
    QMessageBox messageBox;
    messageBox.setIcon(QMessageBox::Information);
    messageBox.setWindowTitle("提示");
    messageBox.setWindowIcon(QIcon(":/images/resources/images/boss.png"));
    messageBox.setText("4*4矩阵的左上角坐标是: (" + sx4.toString() + ", " + sy4.toString() + ")");
    messageBox.setStandardButtons(QMessageBox::Ok);
    messageBox.setDefaultButton(QMessageBox::Ok);

    messageBox.exec();
}

```

效果如下:



• 4.3 排行榜

排行榜显示的核心逻辑是，点击“普通模式”和“比赛模式”可以切换读取的Json 排行榜文件，读取文件并将文件中存储的用户数据使用 QVector 存储：

```

void rank_show:: choose_common()//练习模式排行榜
{
    users=userbase::loadUsersFromJson("./rank/common.json");
}

void rank_show::choose_boss()//比赛模式排行榜
{
    users=userbase::loadUsersFromJson("./rank/boss.json");
}

```

然后点击不同的 pushbutton 进行不同的排序：

```

void rank_show::choose_id()//按id排序
{
    if(issuerempty())return;
    else
    {
        usermodel->clear();
        std::sort(users.begin(),users.end(),[](const userbase&a,const userbase&b){return a.get_id()<b.get_id();});

        show_rank();
    }
}

void rank_show::choose_timeuse()//按最短用时排序
{
    if(issuerempty())return;
    else
    {
        usermodel->clear();
        std::sort(users.begin(),users.end(),[](const userbase&a,const userbase&b)
        {if(a.get_time()!=b.get_time())
        return a.get_time()<b.get_time();
        else return a.get_id()<b.get_id();});

        show_rank();
    }
}

void rank_show::choose_pass_time()//按通关次数排序
{
    if(issuerempty())return;
    else
    {
        usermodel->clear();
        std::sort(users.begin(),users.end(),[](const userbase&a,const userbase&b)
        {if(a.get_pass_time()!=b.get_pass_time())
        return a.get_pass_time()>b.get_pass_time();
        else return a.get_id()<b.get_id();});

        show_rank();
    }
}

```

然后设置由 `QTableView` 管理的 `QStandardItemMode`，在这里，每一个用户数据条目都是一个 `item`，在 `usermodel` 中添加进去并设置相对位置即可显示出来。并且在这里对每个排行条目进行了样式设置，设置了字体和大小，并设置了文本居中。：

```

void rank_show::show_rank()
{
    //设置表头
    userModel->setColumnCount(4);
    userModel->setHeaderData(0,Qt::Horizontal,tr("排名"));
    userModel->setHeaderData(1,Qt::Horizontal,tr("昵称"));
    userModel->setHeaderData(2,Qt::Horizontal,tr("最短用时/秒"));
    userModel->setHeaderData(3,Qt::Horizontal,tr("通关次数"));
    for(int i=0;i<users.size();i++)//逐条信息进行设置
    {
        QStandardItem*rankitem=new QStandardItem(QString::number(i+1));//排名
        rankitem->setData(Qt::AlignCenter, Qt::TextAlignmentRole);
        rankitem->setFont(QFont("黑体",15,QFont::Normal));

        QStandardItem*iditem= new QStandardItem(QString(users[i].get_id()));//用户名id
        iditem->setFont(QFont("黑体",15,QFont::Normal));
        iditem->setData(Qt::AlignCenter, Qt::TextAlignmentRole);

        QStandardItem*timeitem=new QStandardItem(QString::number(users[i].get_time()));//最短时间
        timeitem->setData(Qt::AlignCenter, Qt::TextAlignmentRole);
        timeitem->setFont(QFont("黑体",15,QFont::Normal));

        QStandardItem*passtimeitem=new QStandardItem(QString::number(users[i].get_pass_time()));//通关次数
        passtimeitem->setData(Qt::AlignCenter, Qt::TextAlignmentRole);
        passtimeitem->setFont(QFont("黑体",15,QFont::Normal));

        userModel->setItem(i,0,rankitem);
        userModel->setItem(i,1,iditem);
        userModel->setItem(i,2,timeitem);
        userModel->setItem(i,3,passtimeitem);
    }
}

```

关于 tableview 的样式设置，在这里使用了 stylesheet 函数进行 CSS 样式设置：

```

tableview->setShowGrid(false); // 不显示网格线
tableview->horizontalHeader()->setHighlightSections(false); // 不高亮选中列
tableview->verticalHeader()->setVisible(false); // 隐藏垂直表头（通常包含行号）
tableview->horizontalHeader()->setDefaultSectionSize(192);
tableview->horizontalHeader()->setFont(QFont("黑体",15,QFont::Normal));
tableview->horizontalHeader()->setStyleSheet(R"(
QHeaderView::section {
    background-color: transparent;
    color: #000000;
    text-align: center;
}
)");
tableview->setStyleSheet(
    " background-color: transparent;"
    "border:none;"
);

```

效果如下：

排名	昵称	最短用时/秒	通关次数
1	lsj	9	2
2	lll	11	1
3	ooo	11	2
4	llisu	12	1
5	lxj	14	1
6	ooo'	14	1
7	wcb	15	1
8	lsj】	16	1
9	皇家团	16	1
10	liu	18	1
11	晨曦	18	1

• 4.4 基于 socket 的服务端与客户端单线连接的聊天室

本项目聊天室基于 Tcp 协议进行通信，核心模块为 QTcpSocket。本项目在设计上支持单一客户端与服务端在同一局域网下进行通信。

socket 通信原理：socket 是应用层与 TCP/IP 协议族通信的中间软件抽象层，它是一组接口。在设计模式中，socket 其实就是一个门面模式，它把复杂的 TCP/IP 协议族隐藏在 socket 接口后面，对用户来说，一组简单的接口就是全部，让 socket 去组织数据，以符合指定的协议。

Tcp 通信建立连接的过程，被称为“三次握手”，即客户端向服务器发送一个 SYN J，服务器向客户端响应一个 SYN K，并对 SYN J 进行确认 ACK J+1，客户端再向服务器发一个确认 ACK K+1，完成三次握手之后，客户端和服务端建立起了链接。在 QTcpSocket 中，这个过程被封装为了客户端的 connecttoHost 函数和服务端的 listen 函数，在服务端开始 listen 后，客户端调用 connecttoHost 函数发起连接请求。

而 Tcp 通信连接断开的过程被称为“四次挥手”，具体过程为：某个应用进程首先调用 close 主动关闭连接，这时 TCP 发送一个 FIN M，另一端接收到 FIN M 之后，执行被动关闭，对这个 FIN 进行确认。它的接收也作为文件结束符传递给应用进程，因为 FIN 的接收意味着应用进程在相应的连接上再也接收不到额外数据。一段时间之后，接收到文件结束符的应用进程调用 close 关闭它的 socket。这导致它的 TCP 也发送一个 FIN N，接收到这个 FIN 的源发送端 TCP 对它进行确认。这样每个方向上都有一个 FIN 和 ACK。对于 QTcpSocket，对应的函数是 disconnectFromHost，对于 QTcpServer，对应的函数是 close。

对于建立连接后，聊天室的发送信息的过程，主要模块是 write 函数。当某一端使用了 write 函数发送信息后，该信息被写入了操作系统的写缓冲区，

然后由系统内核读取了写缓冲区的数据后发送给另一端的读缓冲区，另一端调用 read 函数读取数据。

1.客户端

首先发起连接请求，在这里服务器地址需要手动输入：

```
void client_widget::connect_to_server()
{
    QString hostaddress=server_address->text();//获取服务器地址
    quint16 port=server_port->text().toUShort();//获取服务器端口号
    socket->connectToHost(hostaddress,port);//发起连接请求
}
```

在连接成功后会触发槽函数，进行控件权限的设置更改：

```
connect(socket, &QTcpSocket::connected, this, &client_widget::connected_to_server);

void client_widget::connected_to_server()
{
    button_connect->setEnabled(false);//“开始连接”不可在触发
    button_disconnect->setEnabled(true);//设置为可触发
    button_send->setEnabled(true);
    connectionStatusLabel->setText("已连接");//更改状态显示
}
```

断开连接同理，断开后会清空本机聊天框内容：

```
void client_widget::disconnect_from_server()
{
    socket->disconnectFromHost();//断开连接
    communicationLog->clear();//清空聊天框
}

void client_widget::disconnected_from_server()
{
    button_connect->setEnabled(true);
    button_disconnect->setEnabled(false);
    button_send->setEnabled(false);
    connectionStatusLabel->setText("未连接");
}
```

连接成功后，需要进行发送信息，在输入栏输入信息后，点击发送触发槽函数发送信息：

```

void client_widget::send_message()
{
    QString message=message_line->text();//获取信息文本
    if(!message.isEmpty())//信息非空
    {
        QByteArray data=message.toUtf8();//转化为utf-8格式编码
        socket->write(data);//将数据写入系统的写缓冲区
        message_line->clear();
        show_message("我:"+message);//聊天框显示信息
    }
}

void client_widget::show_message(const QString &message)
{
    communicationLog->append(message);//添加信息显示
}

```

在此基础上，添加了回车键的事件过滤器，在 button_send 的 isenabled 值为 true 的情况下按下回车即可发送信息：

```

bool client_widget::eventFilter(QObject *obj, QEvent *event)
{
    if (obj == message_line && event->type() == QEvent::KeyPress)
    {
        QKeyEvent *keyEvent = static_cast<QKeyEvent*>(event);
        if ((keyEvent->key() == Qt::Key_Return || keyEvent->key() == Qt::Key_Enter)&&(button_send->isEnabled()))
        {
            send_message();
            return true;
        }
    }
    return QWidget::eventFilter(obj, event);
}

```

对于接受到来自服务端的信息，当信息被传输到客户端的读缓冲区时，会触发 readyRead 信号，调用槽函数读取数据并显示在聊天框上：

```

void client_widget::readyRead_handler()
{
    while (socket->bytesAvailable() > 0)//缓冲区有数据可读
    {
        QByteArray data = socket->readAll();//读取数据
        QString message = QString::fromUtf8(data);//从utf-8编码转为字符串
        show_message("服务端: " + message);//显示文本信息
    }
}

```

2.服务端

对于服务端，首先需要开始监听可能出现的连接请求：

```

void server_widget::start_Server()
{
    QString hostaddress=host_address->text();//获取本机地址
    quint16 port_=port->text().toInt();//本机端口号
    //监听是否有请求连接本机的客户端，有则更改控件权限
    if(server->listen(QHostAddress(hostaddress), port_))
    {
        button_start->setEnabled(false);
        button_stop->setEnabled(true);
        button_send->setEnabled(true);
        serverStatusLabel->setText("正在监听: " + hostaddress + ":" + QString::number(port_));
    }
    else
    {
        communicationLog->append("无法启动: " + server->errorString());
    }
}

```

其中，对于本机的 ip 获取，设置了专门的函数：首先需要获取系统上所有网络接口的 IP 地址列表，然后遍历这个地址列表，找到第一个非回环 ipv4 地址并返回，找不到则返回 127.0.0.1 作为本机回环地址：

```

QString server_widget::get_localIP_address()
{
    const QList<QHostAddress> &addresses = QNetworkInterface::allAddresses();//获取网络接口地址列表
    for (const QHostAddress &address : addresses)//遍历列表
    {
        if (address.protocol() == QAbstractSocket::IPv4Protocol && address != QHostAddress::LocalHost) {
            return address.toString();//找到第一个非回环IPV4地址并返回
        }
    }
    return "127.0.0.1";//回环地址
}

```

监听到客户端连接请求，需要对请求进行处理，触发槽函数：

```

void server_widget::incomingConnection()
{
    while(server->hasPendingConnections())//有连接请求
    {
        QTcpSocket*clientsocket=server->nextPendingConnection();//创建QTcpSocket对象进入列表管理
        clients.append(clientsocket);
        connect(clientsocket,&QTcpSocket::readyRead,this,&server_widget::readyReadHandler);//处理该链接的读取信息
        connect(clientsocket,&QTcpSocket::disconnected,this,&server_widget::disconnectedHandler);//处理该链接的断连
        show_message("新客户端连接成功: " + clientsocket->peerAddress().toString());
    }
}

```

对于发送信息，大体逻辑与客户端一致，不过需要注意的是，在这里，当服务端发送信息的时候，遍历服务端维护的每个客户端的 QTcpSocket 对象，这个 QTcpSocket 对象是服务端维护的服务端与客户端的特定链接，由这个 QTcpSocket 对象调用 write 函数，将信息写入服务端与该对应客户端的发送缓冲区：

```

void server_widget::send_message()
{
    QString message=message_line->text();//获取信息文本
    if(!message.isEmpty())
    {
        QByteArray data=message.toUtf8();//转为utf-8编码
        for (QTcpSocket *clientSocket : clients)//遍历每个客户端连接
        {
            clientSocket->write(data);//向每个客户端写入数据
        }
        show_message("服务端: " + message);
        message_line->clear();
    }
}

```

在这里，发送信息时服务端与客户端的显示信息函数与回车事件过滤器一致。

当服务端读缓冲区接收到某一客户端发来的信息时，会触发槽函数进行文本显示，在这里 sender() 返回触发 readyread 信号的对象基类 QObject 指针，使用 qobject_cast<QTcpSocket*>(sender()); 将该指针转化为 QTcpSocket 类型的指针，如果转化失败则返回空指针。与 sendmessage 同理在这里 clientsocket 代表服务端维护的服务端与对应客户端的连接，当客户端给服务端发送信息后，该链接的读缓冲区接收到了信息，然后调用 readall 读取信息并显示在服务端的聊天框。：

```

void server_widget::readyReadHandler()
{
    QTcpSocket *clientsocket = qobject_cast<QTcpSocket*>(sender());//指针转化为QTcpSocket类型
    if (!clientsocket)
    {
        return;
    }
    while (clientsocket->bytesAvailable() > 0)//该特定链接的读缓冲区有数据
    {
        QByteArray data = clientsocket->readAll();
        QString message = QString::fromUtf8(data);
        show_message("客户端:"+ message);
    }
}

```

当客户端断开连接后，服务端会删除对应链接：

```

void server_widget::disconnectedHandler()
{
    QTcpSocket *clientsocket = qobject_cast<QTcpSocket*>(sender());//指针转化
    if (!clientsocket)
    {
        return;
    }
    clients.removeOne(clientsocket);//删除列表里的对象，即对应链接
    delete clientsocket;//释放资源
    clientsocket=NULL;
}

```

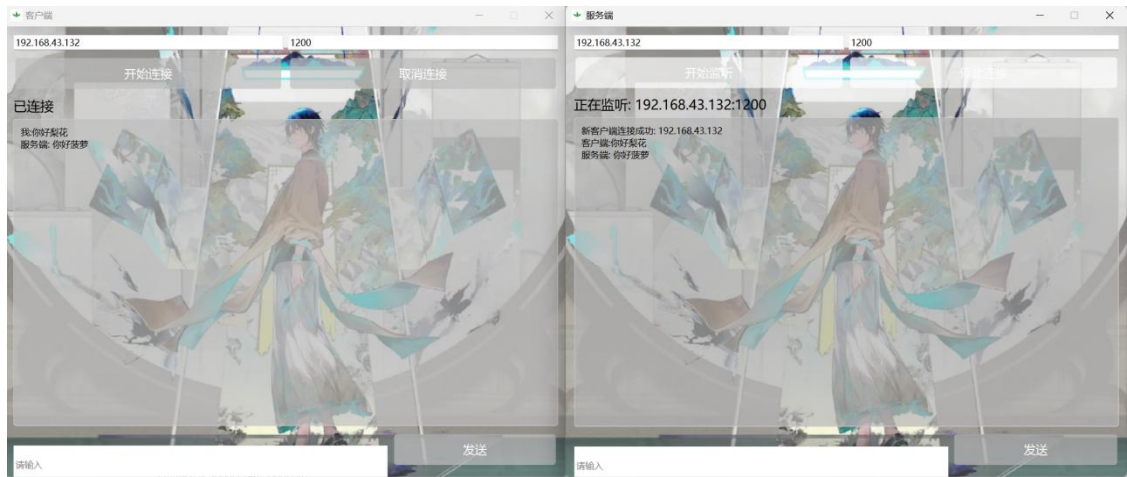
停止监听后更改控件属性并关闭 QTcpServer：


```

void server_widget::stop_Server()
{
    server->close();
    button_start->setEnabled(true);
    button_stop->setEnabled(false);
    button_send->setEnabled(false);
    serverStatusLabel->setText("未开始监听");
    communicationLog->clear();
}

```

最后的通信效果如下：



反思与总结

- 1.本项目代码行数超过 3000 行，但是由于最开始项目架构没有思考清楚，比赛模式和练习模式使用同一套 UI 导致了大量的代码重复，十分冗余，应该在最开始设置一个页面基类，然后再两个模式中继承并扩展，这样子可以减少约六百行代码，大大提高简洁性。
- 2.其实做到中间比赛模式的时候卡了很久，刚开始因为随机生成算法的 `rand()` 函数使用不当导致大量不合法情况从而阻塞了 UI 线程（大约 200-500ms 没有跑出来），后面不想着优化算法，转头去研究起了多线程能否解决算法超时导致的 UI 线程阻塞的问题，但是在这里卡住了一个星期，线程资源泄露的问题一直没有得到解决，后面也就放弃了这个方案，开始重构起随机生成算法。刚开始针对 `rand` 的用法构思了一个每一个位置都在上一个矩阵的位置的小范围区间内随机生成，但是效果也不理想，且随机范围的参数调节过于复杂，也放弃了这个方案。最后想到超时的问题是因为不合法情况过多，想到能否牺牲单次生成的复杂度来换取不合法情况的大量减少，即算法的核心思想从完全随机变成了类搜索问题，枚举区间。最后实现花了一天 debug 了一天问题迎刃而解。

3.关于排行榜的样式设置，刚开始由于对 CSS 不熟悉导致表格的样式设置一直不成功，浪费了很长时间，后来才发现是因为设置对象的某一处语法有多余部分。

4.关于聊天室，这是整个项目最有技术含量也最遗憾的地方，本来最初设想通过联网能够做出玩家间的文件传输，但是由于文件传输方面的封装性不高，需要手动造轮子的地方较多，对计算机网络的知识了解不多，实现起来较为困难，又碍于时间原因只能作罢。此外，虽然聊天室服务端的代码设置是可以访问个客户端，但客户端设置上只能和服务端进行单线通信，没有实现服务端对多个客户端的通信，实属遗憾。

5.虽然本项目的诟病颇多，但是面向项目需求进行学习的过程让我受益匪浅，体会到了开发的漫长和针对某一个模块的难点攻关的艰辛，感受到了在茫茫资料中寻找那一点自己所需要的内容的疲惫，但是解决了问题也让自己的经验增长了，也借此机会学习了部分计算机网络相关的知识，收获颇多。