

一、系统总体介绍

单词消除游戏由两类参与者组成：闯关者（即游戏玩家），出题者（为游戏增加游戏中使用单词）。游戏规则为，游戏每一轮，程序会根据该关卡难度，显示一个单词，一定时间后单词消失。闯关者需要在相应地方输入刚刚显示并消失的单词，如果闯关者输入正确（即闯关者输入的单词与刚刚显示的单词完全一致，包含大小写）则为通过。一关可以由一轮或者多轮组成。

二、开发工具

1.VSCode

2.Qt Creator 10.0.0 (Community)

3.Navicat MySQL 8.0

4.Qt Version 5.15.2

5.CMake 3.25.1

三、设计内容

闯关者属性含有：闯关者姓名、闯关者密码、已闯关关卡数、闯关者经验值、闯关者等级。出题者属性含有：出题者姓名、出题者密码、出题者出题数目、等级。

具体功能包括：

■实现闯关者，出题者本地的注册、登录。

■程序支持多人注册，同一时间只有一人可以登录。

■实现游戏规则：出题者增加游戏中使用单词。游戏每一关，程序会根据该关卡难度，显示一个单词，一定时间后单词消失。闯关者需要在相应地方输入刚刚显示并消失的单词，如果闯关者输入正确则为通过。

■任何角色均可查询所有闯关者、出题者，按照属性查找相应闯关者、出题者。

■可以根据闯关者闯过关卡数、经验、等级等对闯关者排名，根据出题者出题数目、等级对出题者排名。

■每一关的难度要有所增加，体现为如下两个条件：

1、进行轮数增多（即单词数目增加，如：前三关仅仅通过一个单词就过关，后续需要通过两个、三个甚至更多才过关）；

2、单词显示时间缩短（随着关卡的增加显示时间越来越短）。

■闯关者每闯过一关，增加一定经验值。经验值会根据闯过的该关卡的关卡号、该关的闯关耗费时间共同决定。当经验值累计到一定程度闯关者等级增加。闯关失败需要重新闯该关。

■游戏自带词库，而且已经注册的出题者可以为系统出题，即增加词库的新词，已经存在的单词不能再次添加。

■每成功出题一次，更新该出题者的出题数目。出题者等级根据出题人成功出题数目来升级。

■游戏为服务器多人游戏平台，使用客户端/服务器的方式，同一时间可以多人登录系统。

■将所有闯关者、出题者信息保存在服务器。使用 socket 进行通信。客户端可以启动多个同时与服务器交互，服务器具有并发处理能力。

四、算法设计思路

本游戏绝大部分运算均在本地进行，服务端仅进行数据的增删改查。

1.类的设计

(1) 客户端

客户端由多个界面和有关参与者的类组成。

有关参与者的类有三个，分别是 Member，Player 和 Tester。其中 Member 包含两个类的基本属性：用户名、密码、经验和等级。

Player 类包含通关数，Tester 类包含出题数。

其余类分别实现登录、注册、查询、添加单词，单人多人游戏功能。

(2) 服务端

服务端主要是实现 socket 通信、多线程处理 socket 和数据库交互。

2.客户端界面介绍及功能实现

(1)有关参与者的类

由于玩家和出题者均存在用户名，用户密码，经验和等级这四个共同属性。

因此设计 Member 类，Player 类和 Tester 类继承自 Member 类和它们的 Get，Set 方法。

Player 类独有属性通关数，Tester 类独有属性出题数，分别存在 Get，Set 方法。

```
class Member : public QObject
{
    Q_OBJECT
protected:
    QString password; //user's pwd
    double exp; //user's exp
    QString name; //user's name
    int rank; //user's rank(need to be calculated)
public:
    /*constructor*/
    Member();
    Member(QString name,QString password);
    /*Setter*/
    void setPassword(QString password);
    void setExp(double exp);
    void setName(QString name);
    void setRank(int rank);
    /*Getter*/
    QString getPassword();
    double getExp();
    QString getName();
    int getRank();
    /*public method*/
    void addExp(int expNum);
    int calExp(int wordlength);
};
```

```

class Player : public Member
{
    Q_OBJECT
private:
    int passNum; //The number of challenges passed.
public:
    /*constructor*/
    Player(QString username,QString password,int exp,int rank,int passNum);
    Player();
    Player(QString username,QString password);
    /*Setter*/
    void setPassNum(int passNum);
    /*Getter*/
    int getPassNum();
    /*public method*/
    //bool login();
};

```

```

class Tester : public Member
{
    Q_OBJECT
private:
    int quesCreatedNum; //Number of questions created
public:
    /*constructor*/
    Tester(QString username,QString password,int exp,int rank,int quesCreatedNum);
    Tester();
    Tester(QString username,QString password);
    /*Setter*/
    void setQuesCreatedNum(int quesCreatedNum);
    /*Getter*/
    int getQuesCreatedNum();
    /*public methods*/
    //bool login();
};

```

(2)注册登录界面(实现于 widget.h widget.cpp 中)

I .连接服务器

ip 指定为本机 ip: 127.0.0.1

端口为: 8989

成功链接服务器后弹窗提醒。

```

void Widget::connectServer()
{
    m_tcp=new QTcpSocket;
    m_tcp->connectToHost(QHostAddress(ip),port);
    connect(m_tcp,&QTcpSocket::connected,this,&Widget::connectOK);
    connect(m_tcp,&QTcpSocket::disconnected,this,[=]() {
        qDebug()<<"已经与服务器断开连接!";
        m_tcp->close();
        m_tcp->deleteLater();
        emit connectOver();
    });
}

```

```

//connect successfully
connect(this,&Widget::connectOK,this,[=]() {
    QMessageBox::information(this,"连接服务器","已经成功链接服务器, 恭喜!");
});

```

(3)登录功能实现

点击登录按钮,会根据类型发送请求,注册信号,根据服务器端回复的数据来决定操作。服务器端与客户端约定好传输方式:

0 代表失败,1 代表成功,后面是该用户的所有数据。

```

/*login authentication*/
connect(ui->loginButton,&QPushButton::clicked,[this]() {
    QString username=ui->unameText->toPlainText();
    QString password=ui->pwdText->toPlainText();
    QString type=ui->typeCbx->currentText();
    if(type=="玩家")
    {
        QString msg="playerLogin "+username+" "+password;
        m_tcp->write(msg.toUtf8().data());
        //After receiving the information from the server, a ready signal will be sent
        connect(m_tcp, &QTcpSocket::readyRead, this, &Widget::playerLogin);
    }
    else if(type=="出题者")
    {
        QString msg="testerLogin "+username+" "+password;
        m_tcp->write(msg.toUtf8().data());
        //After receiving the information from the server, a ready signal will be sent
        connect(m_tcp, &QTcpSocket::readyRead, this, &Widget::testerLogin);
    }
});

```

为防止内存泄漏,本程序所有页面跳转均采用关闭当前所有界面,而后传递 tcp 参数和用户数据的方式。

```

void Widget::playerLogin()
{
    QString array = m_tcp->readAll();
    qDebug() << "receive player login msg:" <<array;
    QStringList info = array.split(' ');
    if(info.at(0)=="playerLoginBack")
    {
        if(info.at(1)=="0")
            QMessageBox::warning(this, tr("登录失败"), tr("用户名或密码有误或用户已登录"));
        else if(info.at(1)=="1") //username and pwd are correct
        {
            /*jump to player page*/
            QString username=ui->unameText->toPlainText();
            QString password=ui->pwdText->toPlainText();
            Player* player=new Player(username,password,info.at(4).toDouble(),info.at(5).toInt(),info.at(6).toInt());
            PlayerPage *playerPageWidget = new PlayerPage(player,m_tcp);
            playerPageWidget->show();
            this->close();
            QMessageBox::information(this, tr("登录成功"), tr("登录成功"));
            disconnect(m_tcp, &QTcpSocket::readyRead, this, &Widget::playerLogin);
        }
    }
}

```

管理员登录同理：

```

void Widget::testerLogin()
{
    QString array = m_tcp->readAll();
    qDebug() << "receive tester login msg:" <<array;
    QStringList info = array.split(' ');
    if(info.at(0)=="testerLoginBack")
    {
        if(info.at(1)=="0")
            QMessageBox::warning(this, tr("登录失败"), tr("用户名或密码有误或用户已登录"));
        else if(info.at(1)=="1") //username and pwd are correct
        {
            /*jump to player page*/
            QString username=ui->unameText->toPlainText();
            QString password=ui->pwdText->toPlainText();
            Tester* tester=new Tester(username,password,info.at(4).toDouble(),info.at(5).toInt(),info.at(6).toInt());
            testerPage *testerPageWidget = new testerPage(tester,m_tcp);
            testerPageWidget->show();
            this->close();
            QMessageBox::information(this, tr("登录成功"), tr("登录成功"));
            disconnect(m_tcp, &QTcpSocket::readyRead, this, &Widget::testerLogin);
        }
    }
}

```

II.注册功能实现(实现于 registration.cpp registration.h)

点击按钮，跳转到注册界面。采用 QT 独有信号槽机制，当点击确认注册按钮时，向服务器端发送请求。根据服务器端回复的数据来决定操作。服务器端与客户端约定好传输方式：0 代表失败，1 代表成功，然后跳转到登陆后界面。

```

connect(ui->confirmButton,&QPushButton::clicked,this,userRegister);
/*accept server info to send suitable signal*/
connect(tcp, &QTcpSocket::readyRead, this, registration::transPage);

```

```

void registration::userRegister()
{
    QString msg="";
    QString username=ui->unameText->toPlainText();
    QString pwd=ui->pwdText->toPlainText();
    QString type=ui->typeCbx->currentText();
    if(type=="玩家")
        msg+="playerRegistration ";
    else if(type=="出题者")
        msg+="testerRegistration ";
    msg+=username+" "+pwd+" ";
    tcp->write(msg.toUtf8().data());
}

```

```

void registration::transPage()
{
    QString msg = tcp->readAll();
    qDebug() << "receive registration msg:" <<msg;
    QStringList info = msg.split(" ");
    if(info.at(0)=="playerRegistrationRecv")
    {
        if(info.at(1)=="1")
        {
            /*jump to player page*/
            Player* player=new Player(info.at(2),info.at(3),0,0,0);
            playerPage *playerPageWidget = new playerPage(player,tcp);
            playerPageWidget->show();
            this->close();
        }
        else if(info.at(1)=="0")
        {
            QMessageBox::warning(this, tr("注册失败"), tr("用户名重复!"));
        }
    }
    else if(info.at(0)=="testerRegistrationRecv")
    {
        if(info.at(1)=="1")
        {
            /*jump to tester page*/
            Tester* tester=new Tester(info.at(2),info.at(3),0,0,0);
            testerPage *testerPageWidget=new testerPage(tester,tcp);
            testerPageWidget->show();
            this->close();
        }
        else if(info.at(1)=="0")
        {
            QMessageBox::warning(this, tr("注册失败"), tr("用户名重复!"));
        }
    }
}

```

(3)玩家操作界面(实现于 playerPage.cpp playerPage.h 中)

The screenshot shows a Qt window titled "Form" with a light gray background. At the top center, the title "玩家" (Player) is displayed in a bold, black, serif font. Below the title, there are four rows of text, each consisting of a label, a value, and a button:

- Row 1: "用户名: user1" followed by a button labeled "查询" (Query).
- Row 2: "经验值: 7" followed by a button labeled "对战" (Battle).
- Row 3: "等级: 6" followed by a button labeled "开始游戏" (Start Game).
- Row 4: "已闯关卡数: 20" followed by a button labeled "退出" (Exit).

The labels and values are in a black, serif font. The buttons are rectangular with a light gray background and a thin black border.

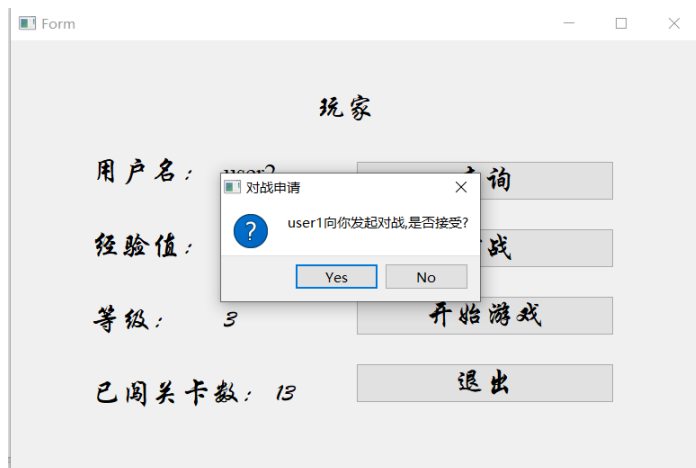
在玩家操作界面可以选择，查询，对战，单人游戏，退出功能和玩家信息查看，通过跳转界面实现。值得注意的是退出按钮，当退出该界面时，会跳转到登录界面，但是创建登陆界面时会新建一个 socket 链接。因此需要在退出程序时关闭已经创建的链接。先向服务器发送一个信号，代表该用户已经退出，以来更新数据表。

```
/*game start*/
connect(ui->playButton,&QPushButton::clicked,[this,m_tcp,playery]() {
    gamePage *gamePageWidget=new gamePage(playery,m_tcp);
    disconnect(tcp,&QTcpSocket::readyRead, this, playerPage::isBattle);
    gamePageWidget->show();
    this->close();
});
/*jump to login page*/
connect(ui->exitButton,&QPushButton::clicked,[this,m_tcp]() {
    Widget *widget=new Widget();
    widget->show();
    disconnect(tcp,&QTcpSocket::readyRead, this, playerPage::isBattle);
    QString msg="playerQuit "+player.getName();
    m_tcp->write(msg.toUtf8().data());
    m_tcp->close();
    m_tcp->deleteLater();
    this->close();
});
/*jump to search page*/
connect(ui->searchButton,&QPushButton::clicked,[this,m_tcp,playery]() {
    searchPage* searchPageWidget=new searchPage(0,m_tcp,playery);
    disconnect(tcp,&QTcpSocket::readyRead, this, playerPage::isBattle);
    searchPageWidget->show();
    this->close();
});
/*ready to battle*/
connect(ui->battleButton,&QPushButton::clicked,[this,m_tcp,playery]() {
    battleSelectPage* battleSelectPageWidget=new battleSelectPage(playery,m_tcp);
    disconnect(tcp,&QTcpSocket::readyRead, this, playerPage::isBattle);
    battleSelectPageWidget->show();
    this->close();
});
```

I. 处理对战功能

先将信号与槽相连接，但退出该界面后，需要将所有链接断开，防止不同的信号抢夺同一条服务器发的信息。

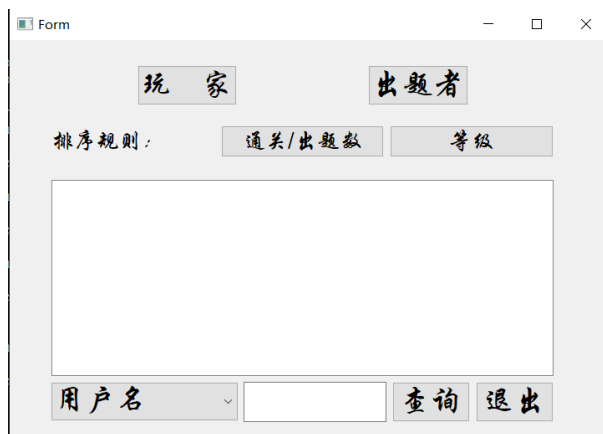
```
connect(m_tcp,&QTcpSocket::readyRead, this, playerPage::isBattle);
```

当收到服务器邀请对战, 取下发起挑战人的信息, 弹窗提醒。如果同意进行对战, 则向服务器发送同意对战信息, 并跳转到对战界面。如果拒绝对战, 则向服务器发送拒绝对战信息。

```
void playerPage::isBattle()
{
    QString msg = tcp->readAll();
    QStringList strList = msg.split(" ");
    qDebug() << "receive isBattle Msg: " << msg;
    if(strList.at(0) == "battleReady")
    {
        QMessageBox::StandardButton reply;
        reply = QMessageBox::question(this, "对战申请", (strList.at(1) + "向你发起对战, 是否接受?"), QMessageBox::Yes | QMessageBox::No);
        if (reply == QMessageBox::Yes)
        {
            disconnect(tcp, &QTcpSocket::readyRead, this, playerPage::isBattle);
            QStringList strList = msg.split(" ");
            QString res = "battleStart " + strList.at(1) + " " + player.getName();
            qDebug() << "send isBattle Msg: " << res;
            tcp->write(res.toUtf8().data());
            battlePage* battlePageWidget = new battlePage(0, strList.at(1), &player, tcp);
            battlePageWidget->show();
            this->close();
        }
        else if (reply == QMessageBox::No)
        {
            QString res = "battleRefused " + strList.at(1);
            tcp->write(res.toUtf8().data());
        }
    }
}
```

II. 查询界面(实现于 searchpage.h searchpage.cpp)



可以查询所有玩家和出题者的信息,并根据排序规则对所有玩家或出题者进行重新排序。实现了根据用户名、经验、等级进行单\多个用户查询,详见具体代码。

查询原理较为简单,根据不同的查询规则,向服务器发送不同的查询申请。根据服务器返回的信息,发送不同的信号,并创建表格。

<pre>//send tcp request void sendSearchInfo(); void sendPlayerInfo(); void sendTesterInfo(); void sendSortByNum(); void sendSortByRank();</pre>	<pre>signals: void playerInfo(QString msg); void testerInfo(QString msg); void searchInfo(QString msg); void SortByNumInfo(QString msg); void SortByRankInfo(QString msg);</pre>	<pre>//receive tcp messages void setSearchInfo(QString msg); void setPlayerInfo(QString msg); void setTesterInfo(QString msg); void setSortByNum(QString msg); void setSortByRank(QString msg);</pre>
---	--	---

链接所有所需的信号。在退出该界面后,仅仅需要关闭 tcp 链接即可。

```
/*send request to server*/
connect(ui->playerButton,&QPushButton::clicked,this,searchPage::sendPlayerInfo);
connect(ui->testerbutton,&QPushButton::clicked,this,searchPage::sendTesterInfo);
connect(ui->searchButton,&QPushButton::clicked,this,searchPage::sendSearchInfo);
connect(ui->numButton,&QPushButton::clicked,this,searchPage::sendSortByNum);
connect(ui->rankButton,&QPushButton::clicked,this,searchPage::sendSortByRank);
/*accept server info to send suitable signal*/
connect(tcp, &QTcpSocket::readyRead, this, searchPage::sendSignal);
/*deal with different signals*/
connect(this,searchPage::playerInfo,this,searchPage::setPlayerInfo);
connect(this,searchPage::testerInfo,this,searchPage::setTesterInfo);
connect(this,searchPage::searchInfo,this,searchPage::setSearchInfo);
connect(this,searchPage::SortByNumInfo,this,searchPage::setSortByNum);
connect(this,searchPage::SortByRankInfo,this,searchPage::setSortByRank);

connect(ui->exitButton,&QPushButton::clicked,[this,m_tcp,type,player,tester]() {
    disconnect(tcp, &QTcpSocket::readyRead, this, searchPage::sendSignal);
    if(type==0)
    {
        /*jump to player page*/
        playerPage *playerPageWidget = new playerPage(player,m_tcp);
        playerPageWidget->show();
        this->close();
    }
    else if(type==1)
    {
        /*jump to tester page*/
        testerPage *testerPageWidget=new testerPage(tester,m_tcp);
        testerPageWidget->show();
        this->close();
    }
});
```

III. 玩家游戏界面(实现于 gamepage.cpp gamepage.h)

玩家游戏界面通过向服务器发送请求, 获取需要记忆的单词, 而后在客户端进行本地运算, 计算出单词的难度, 记忆的时间, 答对后获取的经验以及每一关需要正确拼写单词的数量。以下是计算公式:

$$\text{单词难度: } difficult = \left\lfloor 1.5 + \frac{\text{word.length} - 3}{2} \right\rfloor$$

$$\text{记忆时间: } interval = \frac{difficult}{1 + \text{当前等级} * 0.2} * 1000$$

$$\text{获取经验: } getExp = \lfloor 1.5 + \log_2(\text{word.length}) * 10 \rfloor$$

$$rankNum = \begin{cases} 1 & nowRank = 1 \\ nowRank - 1 & nowRank \% 2 = 1 \\ nowRank & nowRank \% 2 = 0 \end{cases}$$

通关需要单词数:

游戏实现流程如下:

① 游戏初始化, 向服务器发送需要一个单词的请求

```
void gamePage::initGame()
{
    /*ban textline and confirm button*/
    ui->answerEdit->setEnabled(false);
    ui->submitButton->setEnabled(false);
    ui->timeLimitBar->setValue(100);
    ui->answerEdit->setText("");
    QString msg="singalGameWord "+ QString::number(nowRank);
    qDebug()<<"send singalGameWord msg:"<<msg;
    tcp->write(msg.toUtf8().data());
}
```

②收到单词后进行上述计算，然后加载进度条提示用户剩余记忆时间。存在两个计时器：processBarTimer 和 memoryTimer。当 memoryTimer 计时结束后，单词消失，用户可以输入记忆的单词。processBarTimer 将记忆时间均分，每次计时结束均匀递减。

```
void gamePage::gameStart()
{
    QString array = tcp->readAll();
    qDebug() << "receive singalGameWord msg:" <<array;
    QStringList info = array.split(' ');
    if(info.at(0)=="singalGameWordRecv")
    {
        word=info.at(1);
        ui->wordShowLabel->setText(word);
        int difficuty=(int)((1.5+(word.length()-3)/2));
        int interval=(difficuty/(1+nowRank*0.2))*1000;
        qDebug()<<interval;
        processBarTimer=new QTimer(this);

        /*a progress bar display with a minor bug*/
        processBarTimer->start(interval/50);
        connect(processBarTimer, &QTimer::timeout, [=]() {
            int value = ui->timeLimitBar->value();
            if (value <= ui->timeLimitBar->minimum()) {
                processBarTimer->stop();
                return;
            }
            ui->timeLimitBar->setValue(value - 1);
        });

        memoryTimer=new QTimer(this);
        memoryTimer->start(interval*2);
        /*when the memorization time is up*/
        connect(memoryTimer,&QTimer::timeout,[this]() {
            ui->answerEdit->setEnabled(true);
            ui->submitButton->setEnabled(true);
            ui->wordShowLabel->setText("");
            ui->timeLimitBar->setValue(0);
            memoryTimer->stop();
        });
    }
}
```

③当点击提交按钮时，进行判断：输入正确则回到步骤①，否则弹窗提示，重新开始这一关。已经获得的经验和等级重置。

```

void gamePage::submit()
{
    QString answer=ui->answerEdit->text();
    if(answer==word)//update player info in database and get a new word
    {
        finishedWord++;
        player.addExp(player.calExp(word.length()));
        player.setPassNum(player.getPassNum()+1);
        /*update now rank*/
        if(finishedWord==rankNum())
        {
            finishedWord=0;
            nowRank++;
            pastExp=player.getExp();
            pastRank=player.getRank();
        }
    }
    else//player challenge the same rank again
    {
        QMessageBox::warning(this, tr("答案错误"), tr("重新开始该关卡"));
        finishedWord=0;
        player.setExp(pastExp);
        player.setRank(pastRank);
    }
    /*update player's info*/
    ui->username->setText(player.getName());
    ui->ranklevel->setText(QString::number(nowRank));
    ui->expNum->setText(QString::number(player.getExp()));
    ui->rankNum->setText(QString::number(player.getRank()));
    ui->passLevel->setText(QString::number(player.getPassNum()));
    /*start a new game*/
    initGame();
}

```

④当玩家打算退出游戏后，需要对所有数据进行更新。

```

connect(ui->exitbutton,&QPushButton::clicked,[this,m_tcp](){
    disconnect(tcp,&QTcpSocket::readyRead,this,gamePage::gameStart);
    /*update tester info in server*/
    QString msg="playerUpdate "+player.getName()+" "+QString::number(pastExp)+" "+QString::number(pastRank)+" "+QString::number(player.getPassNum());
    tcp->write(msg.toUtf8().data());
    /*jump to player page*/
    player.setExp(pastExp);
    player.setRank(pastRank);
    playerPage *playerPageWidget = new playerPage(&player,m_tcp);
    playerPageWidget->show();
    this->close();
});

```

IV. 双人对战模式

①点击对战按钮，首先进入挑选玩家界面。向服务器发送请求，获取当前在线所有玩家，并以表格形式展现。当选中玩家后，点击确认按钮将向服务器发送请求对战要求。服务器向对手发送邀请，如果双方均同意则跳转到对战界面(实现于 battleselectpage.cpp battleselectpage.h)。

Form

对战模式

请挑选你的对手：

	玩家	经验	等级	通关数
1	user2	89	3	13

确认
退出

```
/*accept server info to send suitable signal*/
connect(m_tcp, &QTcpSocket::readyRead, this, battleSelectPage::setSignals);
connect(this, battleSelectPage::recvInfo, this, battleSelectPage::setPlayerInfo);
connect(this, battleSelectPage::battleBeginSignal, this, battleSelectPage::battleBegin);
connect(ui->confirmButton, &QPushButton::clicked, this, readyBattle);
connect(ui->exitButton, &QPushButton::clicked, [this, m_tcp, playery]() {
    /*jump to player page*/
    disconnect(m_tcp, &QTcpSocket::readyRead, this, battleSelectPage::setSignals);
    playerPage *playerPageWidget = new playerPage(playery, m_tcp);
    playerPageWidget->show();
    this->close();
});
```

由于需要接受服务器发回的不同数据，因此设计该方法，根据不同数据发出不同信号。

```
void battleSelectPage::setSignals()
{
    QString msg = tcp->readAll();
    QStringList strList = msg.split("\n");
    qDebug() << "receive onlinePlayer msg:" << msg;
    if(strList.at(0)=="playerOnlineRecv")
        emit recvInfo(msg);
    else if(strList.at(0)=="battleStart")
        emit battleBeginSignal(msg);
    else if(strList.at(0)=="battleRefused")
    {
        QMessageBox::warning(this, tr("对战失败"), tr("对方拒绝与你战斗!"));
        ui->exitButton->setEnabled(true);
    }
}
```

```

void battleSelectPage::readyBattle()
{
    // 获取当前选中的行号
    QModelIndexList selectedRows = ui->tableWidget->selectionModel()->selectedRows();
    if (selectedRows.isEmpty())
        QMessageBox::warning(this, tr("对战失败"), tr("请选择挑战对手!"));
    else
    {
        QModelIndex cellIndex = ui->tableWidget->model()->index(selectedRows.at(0).row(), 0);
        QString value = cellIndex.data().toString()+" ";
        QString msg="ballteReady "+player->getName()+" "+value;
        tcp->write(msg.toUtf8().data());
        ui->exitButton->setDisabled(true);
    }
}

/*jump to battle page*/
void battleSelectPage::battleBegin(QString msg)
{
    QStringList strList = msg.split("\n");
    battlePage* battlePageWidget=new battlePage(1,strList.at(2),player,tcp);
    battlePageWidget->show();
    this->close();
    disconnect(tcp, &QTcpSocket::readyRead, this, battleSelectPage::setSignals);
}

```

②对战模式双方需要对同一个单词进行回答。(实现于 battlepage.h battlepage.cpp)

y) 因此进行如下设计:

<1> 挑战者首先发送一个需要单词请求, 服务端向两个客户端发送该单词。

```

/*when you jump to this page,game has been started*/
if(type==1)
    initGame();
else
{
    ui->answerEdit->setEnabled(false);
    ui->submitButton->setEnabled(false);
    ui->timeLimitBar->setValue(100);
    ui->answerEdit->setText("");
}

```

<2> 之后比较两人手速, 最先答对的人发送需要单词请求, 更新两人题目。

```

void battlePage::submit()
{
    QString answer=ui->answerEdit->text();
    if(answer==word)//update player info in database and get a new word
    {
        correctWord++;
        initGame();
    }
    else//player challenge the same rank again
        QMessageBox::warning(this, tr("答案错误"), tr("答案错误!"));
}

```

<3>游戏进行 5 轮, 答对题目多者胜。该判断基于本地: 本地存在一个变量 currentWord, 答对单词后自增。当收到五个单词后进入判定。currentWord 小于 3

就会终止游戏，弹出游戏失败指令。

```
else
{
    ui->submitButton->setDisabled(true);
    if(correctWord<3)
        QMessageBox::information(this, tr("对战结果"), tr("你输了!不损失经验"));
    else
        QMessageBox::information(this, tr("对战结果"), tr("你赢了!获得200经验"));
}
```

<4> 之后退出对战模式后更新角色经验等级。

```
if(round-->5)
{
    if(correctWord==3)
    {
        player->addExp(200);
        QString msg="playerUpdate "+player->getName()+" "+QString::number(player->getExp())+" "+QString::number(player->getRank())+" "+QString::number(player->getPassNum());
        tcp->write(msg.toUtf8().data());
    }
    else
    {
        QString msg="battlelose "+player->getName()+" "+opponent;
        tcp->write(msg.toUtf8().data());
    }
}
```

<5> 同时做了逃跑模式，当获得单词数目小于 5 时点击退出视为逃跑。向服务端发送战败信息，服务端接收后向对手发送获胜信息。

```
else if(info.at(0)=="youAreWinner")
{
    QMessageBox::information(this, tr("对战结果"), tr("对手逃跑,你赢得了这场比赛!\n获得200经验"));
    ui->submitButton->setDisabled(true);
    player->addExp(200);
}
```

(4)出题者操作界面(实现于 testerpage.h testerpage.cpp)

出题者查询和玩家查询使用同一界面，不多赘述。仅仅介绍出题界面实现方法。

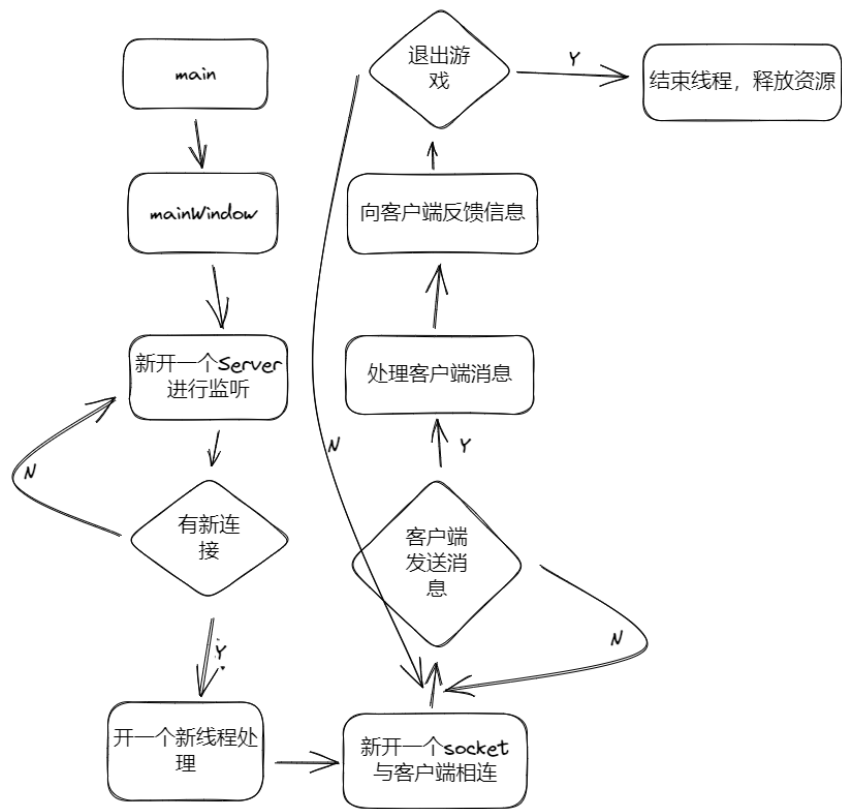
I. 出题界面(实现于 addwordpage.cpp addwordpage.h)

点击提交后将单词和难度发往服务端，接受服务端返回的数据。成功添加更新本地出题者数据，向服务端发送更新数据。

```
void addwordPage::wordRecv()
{
    QString array = tcp->readAll();
    QString word=ui->answerEdit->text();
    qDebug() << "receive word receive msg:" <<array;
    QStringList info = array.split(' ');
    if(info.at(0)=="testerWordRecv")
    {
        if(info.at(1)=="0")
        {
            QMessageBox::warning(this, tr("添加失败"), tr("该单词已存在题库中!"));
        }
        else if(info.at(1)=="1")
        {
            tester.addExp(testExp(word.length()));
            tester.setQuesCreatedNum(tester.getQuesCreatedNum()+1);
            QMessageBox::information(this, tr("添加成功"), tr("添加成功"));
            /*update tester's info*/
            ui->username->setText(tester.getName());
            ui->expNum->setText(QString::number(tester.getExp()));
            ui->rankNum->setText(QString::number(tester.getRank()));
            ui->passLevel->setText(QString::number(tester.getQuesCreatedNum()));
            /*update tester info in server*/
            QString msg="testerUpdate "+tester.getName()+" "+QString::number(tester.getExp())+" "+QString::number(tester.getRank())+" "+QString::number(tester.getQuesCreatedNum());
            tcp->write(msg.toUtf8().data());
        }
        ui->answerEdit->setText("");
    }
}
```

3.服务端功能实现

1.总体架构



2.数据库(实现于 dbUtil.h dbUtil.cpp)

数据表如下设置：


Player:

名	类型	长度	小数点	不是 null	虚拟	键
uname	varchar	255	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	1
pwd	varchar	255	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
exp	double	255	0	<input type="checkbox"/>	<input type="checkbox"/>	
ranker	int	255	0	<input type="checkbox"/>	<input type="checkbox"/>	
passNum	int	255	0	<input type="checkbox"/>	<input type="checkbox"/>	
isOnline	int	255	0	<input type="checkbox"/>	<input type="checkbox"/>	

Tester:

名	类型	长度	小数点	不是 null	虚拟	键
uname	varchar	255	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	1
pwd	varchar	255	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
exp	double	255	0	<input type="checkbox"/>	<input type="checkbox"/>	
ranker	int	255	0	<input type="checkbox"/>	<input type="checkbox"/>	
quesCreatedNum	int	255	0	<input type="checkbox"/>	<input type="checkbox"/>	
isOnline	int	255	0	<input type="checkbox"/>	<input type="checkbox"/>	

Vocabulary:

名	类型	长度	小数点	不是 null	虚拟	键
word	varchar	255	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	 1
difficulty	int	255	0	<input type="checkbox"/>	<input type="checkbox"/>	

这部分主要实现服务器对工程所有的数据的处理,从而实现对客户端的应答,这部分通过 qt 的数据 mysql 接口完成,处理过程就是类似的数据库的管理操作(插入、更新、删除、查找等等),这部分的所有功能全部封装在 dbUtil 这类中。所有函数如下:(相关操作就不再赘述,全是数据库操作)

```
class dbUtil : public QObject
{
    Q_OBJECT
public:
    /*constructor*/
    dbUtil();
    void close();
    /*public method*/
    QString playerRegiste(QString uname,QString pwd);
    QString testerRegiste(QString uname,QString pwd);
    QString playerLogin(QString uname,QString pwd);
    QString testerLogin(QString uname,QString pwd);
    QString allPlayerInfo();           //get all player info
    QString allTesterInfo();          //get all tester info
    QString allSearchUname(int type,QString uname); //according to username and usertype
    QString allSearchNum(int type,int number);      //according to passNum or questionCreatedNum and usertype
    QString allSearchRank(int type,int rank);       //according to rank and usertype
    QString allSortByNum(int type);                 //according usertype
    QString allSortByRank(int type);                //according usertype
    QString allPlayerOnline(QString uname);          You, 昨天 * j ...
    void playerLogout(QString uname);
    void testerLogout(QString uname);
    bool addWord(QString word,int difficult);
    void testerInfoUpdate(int exp,int rank,int quesCreatedNum,QString name);
    void playerInfoUpdate(int exp,int rank,int quesCreatedNum,QString name);
    QString getSignalGameWord(int rank);
private:
    QString hostName="localhost";
    QString dbName="wordMatchGame";
    QString userName="root";
    QString password="wenbo030605";
    QSqlDatabase dbconn;
};
```

3 多线程

服务器端这边: myserver.cpp 会监听是否有客户端登录,如果有的话自动调用这函数: virtual void incomingConnection(qintptr socketDescriptor); 只需要这个函数中为这个客户端新开一个进程就可以实现多客户端模式了,每个客户端都会在单独的进程中运行,互不干扰。

```

void MyServer::incomingConnection(qintptr socketDescriptor)
{
    MyThread *thread = new MyThread(0, socketDescriptor);
    connect(thread, &MyThread::finished, [&]{ qDebug() << "thread is over:"<<QThread::currentThread(); });
    connect(thread, &MyThread::finished, &MyThread::deleteLater);
    thread->start();
}

```

另外处理线程的函数 mythread.cpp，因为每次连接上一个客户端就会开辟一个线程，在新开的线程中都会重写 void MyThread::run()，在里面会新开一个 socket 作为每个客户端通信用的工具：

```

void MyThread::run()
{
    socket = new MySocket(0, this->p);
    connect(socket, &MySocket::disconnected, this, &MyThread::quit, Qt::DirectConnection);
    connect(socket, &MySocket::update_serve, socket, &MySocket::slot_update);
    this->exec();
}

```

4 通信格式

处理通信 socket 的函数是 mysocket.cpp，只要服务器接收到消息就会出发 void MySocket::slot_update(QString msg)，这个更新函数，用来处理“通信格式”：

以下是所有格式截图：

```

void MySocket::slot_update(QString msg, qintptr descriptor)
{
    QStringList info = msg.split(' ');
    qDebug() << "dbcon=new dbUtil()";
    /*scope with player login*/
    if(info.at(0)=="playerLogin") ...
    /*scope with tester login*/
    else if(info.at(0)=="testerLogin") ...
    /*scope with player register*/
    else if(info.at(0)=="playerRegistration") ...
    /*scope with tester register*/
    else if(info.at(0)=="testerRegistration") ...
    /*scope with adding word*/
    else if(info.at(0)=="testerAddWord") ...
    /*scope with search page*/
    else if(info.at(0)=="playerUserName") ...
    else if(info.at(0)=="playerRank") ...
    else if(info.at(0)=="playerNum") ...
    else if(info.at(0)=="testerUserName") ...
    else if(info.at(0)=="testerRank") ...
    else if(info.at(0)=="testerNum") ...
    else if(info.at(0)=="allPlayerInfo") ...
    else if(info.at(0)=="allTesterInfo") ...
    else if(info.at(0)=="playerSortByNum") ...
    else if(info.at(0)=="testerSortByNum") ...
    else if(info.at(0)=="playerSortByRank") ...
    else if(info.at(0)=="testerSortByRank") ...
    /*finish coping with search page*/
    else if(info.at(0)=="singalGameWord") ...
    else if(info.at(0)=="playerOnline") ...
    /*scope with tester info's update*/
    else if(info.at(0)=="testerUpdate")
    {
        dbcon->testerInfoUpdate(info.at(2).toInt(),info.at(3).toInt(),info.at(4).toInt(),info.at(1));
    }
    /*scope with player info's update*/
    else if(info.at(0)=="playerUpdate")
    {
        dbcon->playerInfoUpdate(info.at(2).toInt(),info.at(3).toInt(),info.at(4).toInt(),info.at(1));
    }
    /*scope with player logout*/
    else if(info.at(0)=="playerQuit") ...
    /*scope with tester logout*/
    else if(info.at(0)=="testerQuit")
    {
        dbcon->testerLogout(info.at(1));
    }
    /*deal with battle*/
    else if(info.at(0)=="ballteReady") //this signal is sent by the challenger...
    else if(info.at(0)=="battleStart") //this signal is sent by the challengee...
    else if(info.at(0)=="battleRefused") //this signal is sent by the challengee...
    else if(info.at(0)=="battleGameWord") ...
    else if(info.at(0)=="battleLose") ...
    /*end*/
    delete(dbcon);
    QSqlDatabase::removeDatabase("qt_sql_default_connection");
}

```

为了解释怎样处理通信格式，我举一个简单的例子：比如玩家 A 向玩家 B 发起对战请求。首先在全局 mysocketlist 中找到玩家 B 的 socket，然后跨线程调用 sendData，最终将这些信息写入通信套接字发给客户端。

```
else if(info.at(0)=="ballteReady") //this signal is sent by the challenger
{
    int sig=0;
    //find challengee
    for(int i = 0; i < gamelist.count() ; i++)
    {
        if(gamelist.at(i) == info.at(2))
        {
            sig = i;
            break;
        }
    }
    MySocket *item_battle = mysocketlist.at(sig);
    QString msg="ballteReady "+info.at(1)+" "+info.at(2);
    //item_battle->write(msg.toUtf8().data());
    QMetaObject::invokeMethod(item_battle, "sendData", Qt::QueuedConnection, Q_ARG(QString,msg));
    qDebug()<<"send ballteReady msg:"<<msg<<" My thread:"<<QThread::currentThread();
}
```

五、实验总结

(一) 技术学习

- 1.练习了数据库增删改查
- 2.学习了如何在 qt 框架中使用了 mysql（下载源码编译对应动态链接库）
- 3.学习了 qt 框架的使用，信号/槽机制的练习
- 4.跨线程调用 QTcpSocket::write

(二) 收获

通过这次的程序设计实践，我巩固了许多有关 C++ 的知识，包括类的封装、继承、虚函数、重载等等。这进一步加深了我对“面向对象”编程范式的理解，即通过对特定对象的操作来实现预期的效果。这种编程思想在软件开发中非常重要，因为它通过对象连接程序和信息，从而使程序可以从文件、数据库或其他数据源中获取相关信息，并将其赋值给对象，然后我们可以对这个对象进行操作以实现所需的功能。这种思想使得我们在编写程序、设计软件时更

加清晰地思考，以更高效、更普遍的方式编写出可维护、可扩展、可重用的软件。除此之外，通过这次实践，我也深入了解了客户端-服务器模式的通信以及多线程编程。在这种模式下，客户端向服务器发送请求，服务器作出相应的响应，并通过套接字 `socket` 进行通信。总之，这次实践让我从多个方面提高了编程技能和理解，也让我更加深入地认识了面向对象编程和客户端-服务器通信模式的重要性。