北京郵電大學

《程序设计实践》设计文档

姓　　名：彭　帅

学　　院：计算机学院

专　　业：计算机科学与技术

2019 年 12 月 11 日

目录

规范的软件原型系统

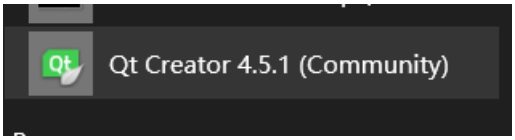
描述：软件原型系统是验证技术可行性的一类重要的应用软件，在新产品研发和科研活动中被广泛应用。它是介于软件单一技术作业和软件产品之间的一种软件形态，对于综合利用程序设计单项技能提出了较高要求。通过本作业，考察学生规范编写代码、合理设计程序、灵活应用技术、协同组织合作等方面的综合能力。

一、系统总体介绍

单词消消乐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

Qt Creator 4.5.1(communtity)



三、设计内容

闯关者属性含有：闯关者姓名、已闯关关卡数、闯关者经验值、闯关者等级；出题者属性：出题者姓名、出题者出题数目、等级，若有需要可以自行添加其余属性。

具体功能包括

（1） 闯关者，出题者本地的注册、登录。
（2） 支持多人注册，同一时间只有一人可以登录。
（3） 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。
（4） 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。

（5） 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。
（6） 闯关者即为游戏玩家，已经注册并登录的玩家可以在系统进行单词消除游戏。每一关的难度要有所增加，体现为如下三个条件中的一个或者多个：1、单词难度可以递增或者持平（即长度加长或不变）； 2、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）； 3、单词显示时间缩短（随着关卡的增加显示时间越来越短）。
（7） 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。
（8） 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（词库中的单词构成一个单词池，但建议根据单词的长度来组织存储。每次出题时，系统从该单词池中按照关卡难度随机的选择相应长度的单词）。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。
（9） 支持多人对战，玩家可以查询其他在线玩家，并对他发起挑战，双方接收之后可以进入对战模式。

设计要点

（1） 采用面向对象的方式，使用类设计；
（2） 在设计类时关注了闯关者、出题者的关联（闯关者与出题者有共同的基类）；
（3） 使用数据库作为存储对象；
（4） 使用 socket 进行通信。
（5） 完成了服务器端客户端程序。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

（一）类的设计

1 客户端

- 实验中我设计了一个基类 Member，包含一些基本属性：用户名、密码、等级、已经闯关数（已经出题数）；然后弄了两个子类 Player 和 Tester
- 登录界面类 LoginDialog（继承 QT 中已有的 QDialog）
- 为了实现完整的功能的玩家游戏界面类 MainWindow 和出题者游戏界面类 MainWindow_tester.（继承 QT 中已有的 QMainWindow）
- 双人对战功能类 Battle

2 服务器

- 数据存储的数据库类 Mysql
- 多线程类 Mythread
- 通信套接字类 Mysocket、监听客户端的类 Myserver
- 主窗口类 MainWindow

（二）客户端界面介绍及功能实现

1 注册登录界面：

1.1 总体介绍

这个类的功能主要是连接服务器，实现玩家和出题者的注册登录功能，主要的函数如下：

```
class LoginDialog : public QDialog
{
    Q_OBJECT
public:
    explicit LoginDialog(QWidget *parent = 0);
    ~LoginDialog();

private slots:
    void on_loginBtn_play_clicked();//玩家登录
    void on_register_play_clicked();//玩家注册
    void on_loginBtn_test_clicked();//出题者登录
    void on_register_test_clicked();//主题者注册
    void link_error();

    //服务器连接slot
    void play_connected_success();//玩家连接服务器
    void test_connected_success();//出题者连接到服务器
    void slot_play_receive();//玩家收到服务器消息
    void slot_test_receive();//出题者收到服务器消息

signals:
    void battle_signal(QString info);//收到对战信号
    void emit_battle(QString info);//发出对战信号
    void ready_signal();//对战就绪信号
    void victory_signal();//对战获胜信号
    void fail_signal();//对战失败信号
    void quit();//结束对战信号

private:
    Ui::LoginDialog *ui;
    int status;
    //服务器
    QHostAddress *serverIP;
};
```

1.2 连接服务器

以玩家为例，出题者类似

当玩家进入到登录界时，创建一个通信用的套接字，然后连接服务器，如果连接成功就进行下面的注册登录功能，这个需要相应的信号槽，比如其中的连接成功的信号槽，收到反馈信息的信号槽。

相应的函数如下：（登录和注册类似）

```
//玩家登录按钮
void LoginDialog::on_loginBtn_play_clicked()
{
    if(tcpsocket == NULL)
    {
        QString ip = "127.0.0.1";
        int port = 8888;
        if(!serverIP->setAddress(ip))//判断是否可以被正确解析
        {
            QMessageBox::warning(this,"Error","无法解析此IP");
            return;
        }
        //创建一个通信套接字，用来和服务器通信
        tcpsocket = new QTcpSocket(this);
        //和服务器进行连接
        connect(tcpsocket, &QTcpSocket::connected, this, &LoginDialog::play_connected_success);
        connect(tcpsocket, SIGNAL(error(QAbstractSocket::SocketError)), this, SLOT(link_error()));
        tcpsocket->connectToHost(*serverIP,port);
        //和服务器连接成功就会出发connected信号
        status = 1;  //登录标志
        //接收到服务器的信息就会出发readyRead信号
        connect(tcpsocket, &QTcpSocket::readyRead, this, &LoginDialog::slot_play_receive);
    }
    else    //第一次密码不对，第二次点击
    {
        if(status == 3 || status == 4)  //避免角色混乱
        {
            QMessageBox::warning(this,"Information","由于之前已经进行了某类操作，所以在未完成之前无法进行下一类操作！");
            return;
        }
        status = 1;
        play_connected_success();
    }
}
```

1.3 注册登录功能

以玩家为例，出题者类似。

当玩家需要登录时，就会给服务器发“登录消息”，服务器收到这个消息之后，会将检索数据库中玩家信息。如果信息存在，就会给客户端这边返回一个“可以登录”信息，否则返回“补课登录”信息；

当玩家需要注册时，先给服务器发“注册消息”，然后也是服务器检索数据库信息，如果“可以注册”，将注册信息写入玩家数据库并返回一个“注册成功”的信息；否则返回“不可注册”的信息。

相应的函数如下：

```
//玩家连接成功
void LoginDialog::play_connected_success()
{
    QString msg ;
    if(status == 1)
       msg = "player_login "+ ui->usrLineEdit->text() +' '+ ui->pwdLineEdit->text();
    else if(status == 2)
       msg = "player_register "+ ui->usrLineEdit->text() +' '+ ui->pwdLineEdit->text();
    //将登录消息发送给服务器
    tcpsocket->write(msg.toUtf8().data());
}
```

1.4 界面截图如下

2 玩家游戏界面

2.1 总体介绍

总的来说就包括单词定时显示和消失、提交单词、查询功能、更新功能、退游等要求，这些功能全部封装在 MainWindow 这个类里面，主要模块如下：

```
class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();
    void Mytimer();
    void ScreenInfo(QTableWidget *source, QString text);

public slots:
    void handleTimeout();//处理单词定时消失槽

private slots:
    void on_submit_clicked();
    void on_Quit_clicked();
    void on_pushButton_player_clicked();//查询玩家信息
    void on_pushButton_tester_clicked();//查询出题者信息
    void on_pushButton_index_clicked();//索引查询

    void on_battle_clicked();  //对战按钮
    void accept_battle(QString);     //接收对战槽
    void make_battle(QString);  //发起对战槽
    void ready_make();  //准备对战槽
    void victory_make();     //对战获胜槽
    void fail_make();    //对战失败槽
    void quit_make();    //退出对战槽

signals:
    void ready_signal();     //准备信号
    void v_signal();     //对战获胜信号
    void f_signal();     //对战失败信号
    void quit_signal(); //对战退出信号

private:
    Ui::MainWindow *ui;
    QTimer *m_pTimer1;
```

2.2 单词显示和消失功能

主要利用 void MainWindow::**Mytimer** 这个函数，首先设置一个初始时间，将 QTimer 的超时信号连接到 void MainWindow::**handleTimeout**()这个槽，当超时之后就是让单词消失，之后如果玩家回答正确就会重新启动计时器，单词重现，这样往复实现游戏要求

相关函数如下：

```
//计时器
void MainWindow::Mytimer() {...}
//超时函数处理单词消失
void MainWindow::handleTimeout()
{
    if(m_pTimer1->isActive()){
        ui->label->hide();
        m_pTimer1->stop();
        m_pTime->start();
    }
}
```

2.3 查询功能

该功能主要利用 qt 中 QTableWidget 类，在玩家和出题者中分别弄了两个按钮：

查询玩家信息：**on_pushButton_player_clicked**();

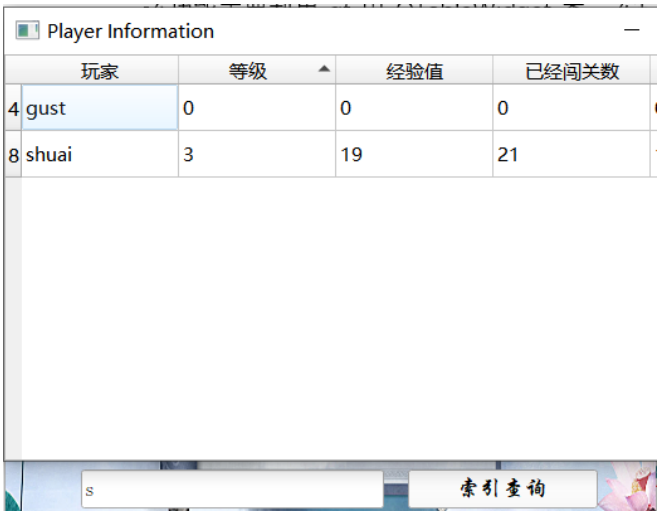
查询出题者信息：**on_pushButton_tester_clicked**();

（以上两个实现思路就是从文件中读取所有玩家或出题者的信息，然后把其写入新建的 QTableWidget 类的对象 table 中，相关的排序功能也只需要调用该类相关函数即可实现）

索引查找：**ScreenInfo**(QTableWidget *source, QString text);

索引查找按钮：**on_pushButton_index_clicked**();

（索引查找就是根据先相关的 index 索引让和这个相关的信息显示在 table 中，而不相关的信息不显示即可）



2.4 对战功能

首先玩家可以查看所有在线玩家，然后对在线玩家发起挑战（挑战信息发给服务器），然后会收到服务器的反馈信息，然后进入到对战界面（之后介绍）

相应的函数如下：

```
//battle 在线玩家
void MainWindow::on_battle_clicked()
{
    QString msg = player.name + " battle " + ui->battlename->text();
    tcpsocket->write(msg.toUtf8().data());
}
```

如果收到“You are battling player”信息：

```
//是否发起battle
void MainWindow::make_battle(QString info)
{
    qDebug() << info;
    QStringList str_info = info.split(' ');
    if(str_info.at(2) == "battling")
    {
        if(QMessageBox::Yes == QMessageBox::information(this,"Battle", info, QMessageBox::Yes |QMessageBox::No))
        else
        {
            qDebug() << "no";
        }
    }
}
```

如果收到“Are you accepting player’s battle”信息：

```
//是否接受battle
void MainWindow::accept_battle(QString info)
{
    qDebug() << info;
    QStringList str_info = info.split(' ');
    if(str_info.at(2) == "accepting")
    {
        if(QMessageBox::Yes == QMessageBox::information(this,"Battle", info, QMessageBox::Yes |QMessageBox::No))
        else
        {
            qDebug() << "no";
        }
    }
}
```

2.5 闯关过程

每过一关单词难度增加我是通过减少单词显示时间和单词长度增加或不变来实现的，该功能封装在提交按钮中：

```
void MainWindow::on_submit_clicked()
```

具体思路就是每次点击这个提交按钮之后，比较显示的单词和玩家提交的单词（**这个都是和服务器交互信息的**），如果正确就会从词库中选择长度增加或者不变的单词作为下一个将要显示的单词（**与此同时更新玩家信息，做到即时更新**），另外我会减小设定的计时器时限让下面单词的显示时间变短。**但是如果闯关失败会自动结束游戏，更新玩家信息之后退出游戏。**

函数处理的主要部分如下：

```
if(ui->answer->text() == ans_word)  //回答正确
{
    ++(player.counted);
    TIME -= 20;//每答一题就会减少显示时间
    int time = m_pTime->elapsed();
    //更新玩家信息
    ui->count->setText(QString::number(player.counted));
    //进度条考虑一下*************************
    //经验值策略
    if(time/1000 >= 5) player.Add_exp(5,ans_word.length()/5);
    else player.Add_exp(time/1000,ans_word.length()/5);
    ui->progressBar->setValue(player.exp);
    //升级策略
    player.Level_up();
    ui->level->setText(QString::number(player.level));
    //*****************************
    msg = "update_player " + player.name +' '+ QString::number(player.level)
        +' '+ QString::number(player.exp) +' '+ QString::number(player.counted);
    tcpsocket->write(msg.toUtf8().data());  //答对一题更新一次
    //*****************************
    //答对之后更换单词
    ans_word = strlist_word.at(player.counted);
    ui->label->setText(ans_word);
    ui->label->show();
    m_pTimer1->start(TIME);
    ui->answer->clear();
}

else    //回答错误
{
    msg = "player_quit " + player.name;
    tcpsocket->write(msg.toUtf8().data());
    //critical信息框需要点击之后才能返回
    QMessageBox::critical(NULL,"提示","GAME OVER");
    close();//最后要结束游戏
}
```

2.6 升级策略

闯关者每闯一关就会根据该关卡的难度和闯关所用时间共同决定经验值 exp 的增加，相应的 exp 增加会让等级也增加。

这个功能封装在 player 类里面：

```
void Level_up() {
    //看经验值每增加5就会升一级
    int i = 0;
    int temp_exp = exp;
    while(temp_exp>=0){
        temp_exp -= 5;
        if(temp_exp>=0) i++;
        else break;
    }
    level = i;
}
void Add_exp(int time,int length) { //增加经验
    exp += 5 - time + length;
}
void Rem_exp(int time,int length) { //减少经验（battle过程中使用）
    exp -= 5 - time + length;
}
```

2.7 游戏界面

首先我一进入界面会把玩家当前信息显示在如下图中的相应位置，然后根据游戏的进行实时更新，那个排行榜功能包含在查询信息按钮里面）

3 出题者游戏界面

3.1 总体介绍

出题者游戏界面功能全封装在 MainWindow_tester 这个类中，主要就是包括增加游戏单词、退游、查询信息、索引等等（其中有几个和玩家游戏界面类的功能是一样），如下截图：

```
class MainWindow_test : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow_test(QWidget *parent = 0);
    ~MainWindow_test();
    void ScreenInfo(QTableWidget *source, QString text);

private slots:
    void on_submit_clicked();//出题按钮
    void on_quit_clicked();//退游按钮
    void on_pushButton_player_clicked();//闯关者信息查询
    void on_pushButton_tester_clicked();//出题者信息查询
    void on_pushButton_index_clicked();//索引查询

private:
    Ui::MainWindow_test *ui;
    QStringList strlist_word;
    QTableWidget *table;
};
```

3.2 增加游戏词库

主要是利用 void MainWindow_tester::**on_submit_clicked**()这个函数，当出题者点击之后，程序会将出题者输入的单词与游戏词库匹配如果单词已经存在就不会加入词库，单词不存在（是新单词）就会加入到词库（数据库）(**“单词存不存在”信息是服务器反馈过来的，如果单词不存在服务器就需要更新词库**)

主要函数如下：

```
//提交单词按钮
void MainWindow_test::on_submit_clicked()
{
    QString text = ui->lineEdit->text();
    if(text == "")
        QMessageBox::critical(this,tr("提示"),tr("请输入单词"));
    else
    {
        QString msg = "submit_word " + text;
        tcpsocket->write(msg.toUtf8().data());

        //让它停在这里几分钟(程序还在进行)，得到更新后的submit
        QEventLoop eventloop1;
        QTimer::singleShot(500,&eventloop1,SLOT(quit()));
        eventloop1.exec(0);

        if(submit == 1) //单词不存在就要更新词库
        {
            ++(tester.counted);
            tester.Level_up();//出题者升级策略
            ui->count->setText(QString::number(tester.counted));
            ui->level->setText(QString::number(tester.level));
            //*****************************
            msg = "update_tester " + tester.name +' '+QString::number(tester.counted)
                    +' '+QString::number(tester.level);
            tcpsocket->write(msg.toUtf8().data());  //出题成功就更新一次
            //*****************************
            ui->lineEdit->clear();
            submit = -1;
        }
        else if(submit == 0)
            QMessageBox::information(this,tr("提示"),tr("单词已存在"));
    }
}
```

3.3 查询功能

该功能和之前玩家界面的查询功能相同，这里不再赘述。
下面展示截图(点击那个三角符号会根据相应属性对玩家排名，如果要索引查找就要输入相关内容)

	玩家	等级	经验值	已经闯关数	is land
1	shuai	3	19	21	0
2	bupt	2	14	8	0
3	ai	2	11	13	0
4	peng	2	10	6	0
5	bi	0	2	1	0
6	gust	0	0	0	0
7	mircale	0	0	0	0
8	putty	0	0	1	0

3.4 升级策略

出题者每次成功出三题，等级就会升高，这个增加策封装在 tester 这个类中：

```
class Tester:public Member
{
    Q_OBJECT
public:
    Tester() {}
    ~Tester() {}
    void Level_up() {
        //看答题数每增加3就会升一级
        int i = 0;
        int temp_count = counted;
        while(temp_count>=0){
            temp_count -= 3;
            if(temp_count>=0) i++;
            else break;
        }
        level = i;
    }
};
```

3.5 游戏界面

界面相比于玩家界面跟简单

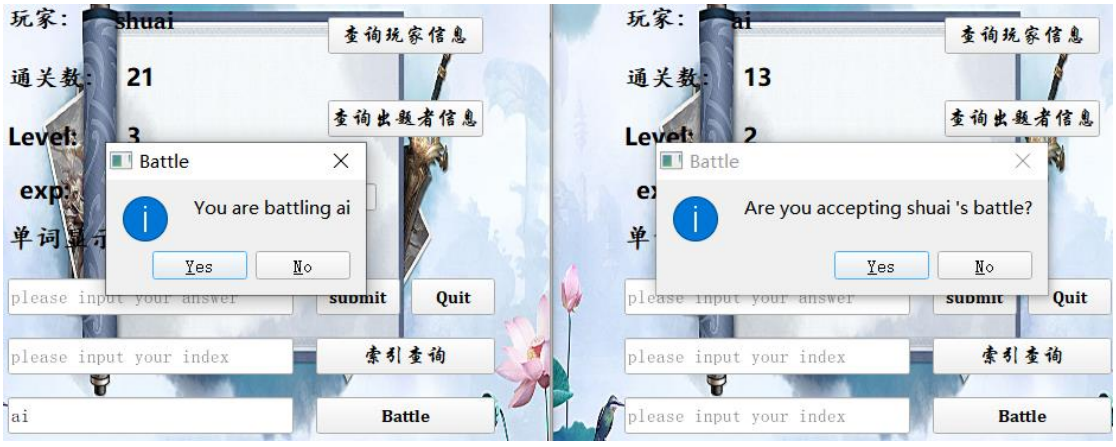
4 玩家对战界面

4.1 进入对战

首先是进入对战状态（这个功能在玩家游戏界面的对战功能），之后进入到对战单独的界面类。这之前需要绑定相应的信号槽，这个很重要，对战的实现就是通过这个和服务器交互的，更确切的说是通过信号槽可以及时相应服务器的回馈信息，其中相关的信号槽如下：

这个信号槽是在玩家界面类生成的，其中管理 battle 双方信号槽就是当有一个人退出 battle 时，另一个人无法继续 battle

```
battle = new Battle(player.name, str_info.at(3));
battle->show(); //控制battle的信息槽
connect(this, SIGNAL(ready_signal()), battle, SLOT(ready()));//battle就绪信号槽
connect(this, SIGNAL(v_signal()), battle, SLOT(victory()));//battle胜利信号槽
connect(this, SIGNAL(f_signal()), battle, SLOT(fail()));//battle失败信号槽
connect(battle, SIGNAL(exit_signal()), this, SLOT(close()));//battle退出信号槽
connect(this, SIGNAL(quit_signal()), battle, SLOT(quit_info()));//管理battle双方信号槽
```



4.2 对战过程

对战过程所用到的函数全部封装在 Battle 这个类里面，其中信息如下：

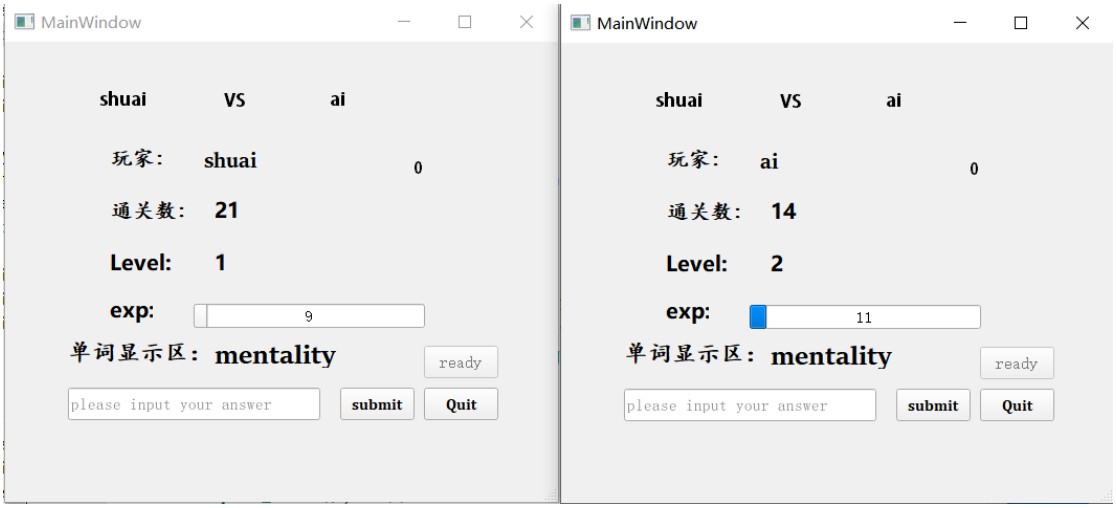
```
class Battle : public QMainWindow
{
    Q_OBJECT

public:
    explicit Battle(QString plA, QString plB, QWidget *parent = 0);
    ~Battle();
    void make_word();    //产生单词

private slots:
    void on_ready_clicked();     //准备就绪按钮
    void on_submit_clicked();    //答题按钮
    void handleout();    //单词出现和消失的槽函数
    //信号槽机制
    void ready();    //准备就绪
    void victory(); //胜利
    void fail();     //失败
    void quit_info();    //退出

    void on_Quit_clicked(); //结束battle

signals:
    void exit_signal();//结束battle信号

private:
    Ui::Battle *ui;
    QTime *Time;     //battle过程计时
    QTimer *m_pTimer;    //battle过程控制单词消失和出现
    int t;  //答出单词用时
};
```

大致过程就是，首先到达 battle 时，battle 双方需要点击 ready 信号告知服务器端自己已经准备就绪，当服务器收到双方 ready 信息时就可以发出 battle 开始的信号（这个功能在接下来的服务器端展示），然后双方会收到信息，然后开始对战，此时双方屏幕上会出现相同的单词并且在一段时候消失，双方需要答出该单词（**答题按钮**），并且会记录时间，然后服务器会收到双方答题所用时间，判断之后给出结果，时间短者获胜并获得相应的经验加成，时间多者或者答题错误都会扣除相应的经验。

每 battle 一次后，双方如果想要进入下一次 battle，同样需要点击 ready 按钮告知服务器（同理），这使得双方都有时间缓冲一下。

对战过程如果有一个玩家退出，另一个玩家就无法进行游戏，必须退出（这是一个符合常规的设计，虽然有点鸡肋）

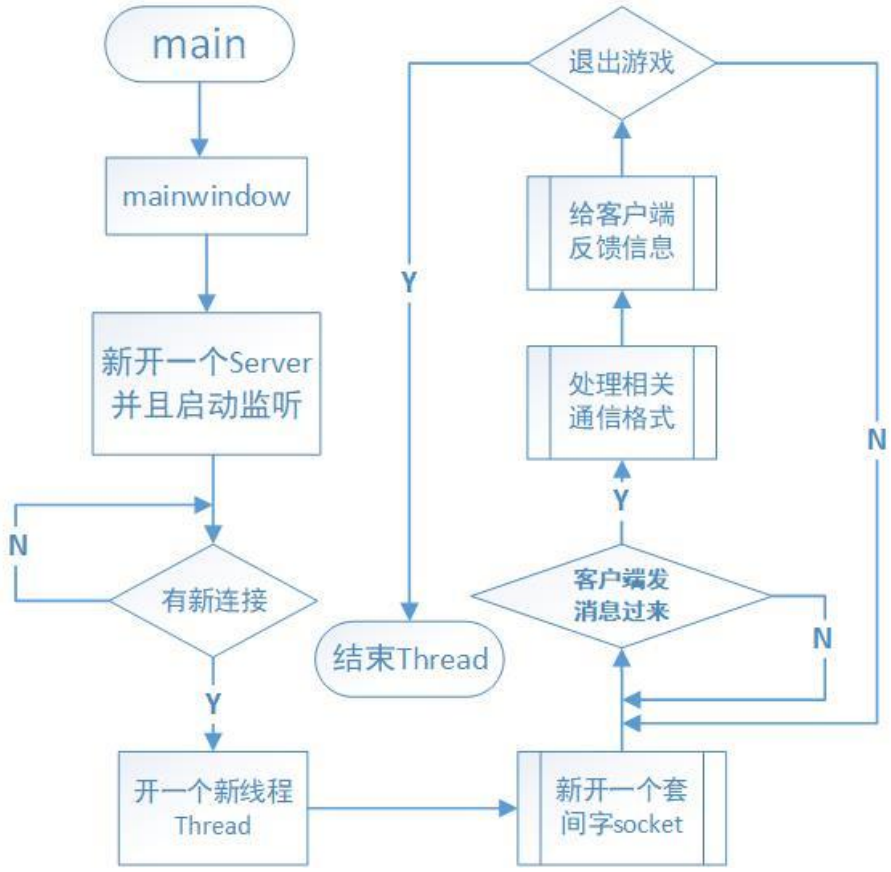
4.3 对战界面


MainWindow
shuai VS ai
玩家： shuai 0
通关数： 21
Level: 1
exp: 9
单词显示区： mentality
ready
please input your answer
submit
Quit
MainWindow
shuai VS ai
玩家： ai 0
通关数： 14
Level: 2
exp: 11
单词显示区： mentality
ready
please input your answer
submit
Quit

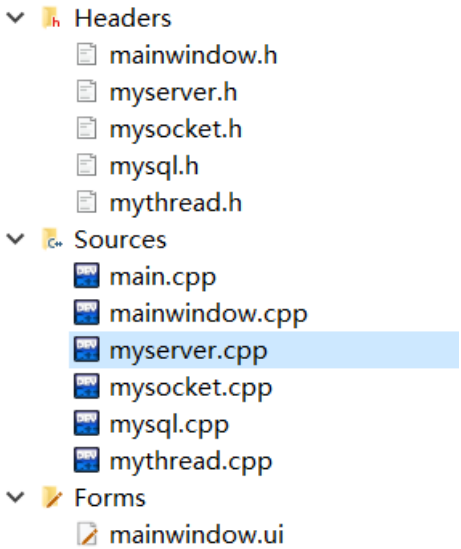

MainWindow
shuai VS ai
玩家： ai 13000
Quit
对方已经退出,您无法继续游戏，请退出!
Yes
No
单词显示区：
ready
please input your answer
submit
Quit

（三）服务器端功能实现

1 总体架构



主要函数如下：



2 数据库

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 Mysql 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 Mysql: public QObject
{
    Q_OBJECT
public:
    Mysql();     //连接数据库
    ~Mysql();
    //玩家
    QString get_player_entry(QString name);//获取某行玩家信息
    QString get_player();
    bool same_name_player(QString name);//是否已经登录
    void update_player_island(QString, QString);
    void update_player(QString, QString, QString, QString);//更新玩家登录信息
    void insert_player(QString, QString);//添加玩家信息
    //出题者
    QString get_tester_entry(QString name);//某行出题者信息
    QString get_tester();
    bool same_name_tester(QString name);
    void update_tester_island(QString, QString);//更新出题者登录信息
    void update_tester(QString, QString, QString);
    void insert_tester(QString, QString);//添加出题者信息
    //单词
    QString get_word_asc();//获取排列好了的单词
    bool same_word(QString word);
private:
    QSqlDatabase db;
};
```

3 多线程

服务器端这边：myserve.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();
    //qDebug() << QThread::currentThread();
}
```

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”：比如接收到 update_player,就会更新玩家信息；接收到 check_player,就会查询玩家信息并且给客户端发送过去。以下是所有格式截图：

```
    qDebug() << msg << descriptor;
    QStringList str_info = msg.split(' ');
    //处理玩家登录信息********************
    if(str_info.at(0) == "player_login") {...}
    //处理玩家注册信息******************
    else if(str_info.at(0) == "player_register") {...}
    //处理出题者登录信息********************
    else if(str_info.at(0) == "tester_login") {...}
    //处理出题者注册信息********************
    else if(str_info.at(0) == "tester_register") {...}
    //处理单词请求**********************
    else if(str_info.at(0) == "request_word") {...}
    //处理更新玩家请求******************
    else if(str_info.at(0) == "update_player") {...}
    //处理玩家退出或者游戏结束行为********************
    else if(str_info.at(0) == "player_quit") {...}
    //查询玩家请求**********************
    else if(str_info.at(0) == "check_player") {...}
    //查询出题者请求************************
    else if(str_info.at(0) == "check_tester") {...}
    //提交单词请求************************
    else if(str_info.at(0) == "submit_word") {...}
    //处理更新出题者请求********************
    else if(str_info.at(0) == "update_tester") {...}
    //处理出题者退游************************
    else if(str_info.at(0) == "tester_quit") {...}

    //battle就绪************************
    else if(str_info.at(0) == "ready") {...}
    //battle结果**********************
    else if(str_info.at(0) == "battle_res") {...}
    //有一个退出时
    else if(str_info.at(0) == "battle_quit") {...}
    //battle提出************************
    else if(str_info.at(1) == "battle") {...}
```

为了解释怎样处理通信格式，我举一个例子：比如说玩家登录时，首先服务器会收到玩家的用户名和密码，然后服务器检索数据库，查看里面有没有该玩家的信息，相应的会有三种情况：11—用户名不存在；00—该用户已经登录；10—用户名或密码错误。最终将这些信息写入通信套接字发给客户端。

```
//处理玩家登录信息********************
if(str_info.at(0) == "player_login")
{
    QString s = "login_back ";
    if(sql.get_player_entry(str_info.at(1)) == "    ")
    {
        s += "11";  //11--用户名不存在
        item->write(s.toUtf8().data()); //写入管道
        return;
    }
    QStringList entry = sql.get_player_entry(str_info.at(1)).split(' ');
    if(str_info.at(2) == entry.at(0))
    {
        if(entry.at(4) == "1")
            s += "00";  //00--已经登录
        else    //01--登录授权
        {
            mysocketlist.append(tcpclientsocketlist.at(i));//登录成功就加入玩家列表
            labellist.append(descriptor);
            gamelist.append(str_info.at(1));
            qDebug() <<"online_player_count: " <<gamelist.count();
            s += "01 " + entry.at(1) + ' ' + entry.at(2) + ' ' + entry.at(3);
            sql.update_player_island(str_info.at(1), "1");  //登录状态
        }
    }
    else
        s += "10";  //10--用户名或密码错误
    item->write(s.toUtf8().data()); //写入管道
}
```

五、实际效果

前面已经叙述了所有的功能，让我们来想象一下游戏过程：

首先打开了服务器，然后启动了四个客户端，有三个登录闯关者，一个登录了出题者(如图 1)，现在每个客户端就能查看到有哪些人在线（查看 is_land）（如图 0）（当玩家登陆之后，服务器会把玩家登录的标志 is_land 置 1，为了不让同一个玩家同时登陆）；

玩游戏时候，玩家信息随时更新，只要玩家做出了合乎游戏规范的动作，这样出题者这时候提供的单词也可能成为某个玩家的下一关，做到实时游戏。（个人觉得这样设计还是挺好的！）

最后退出游戏时，服务器将相关的信息改变

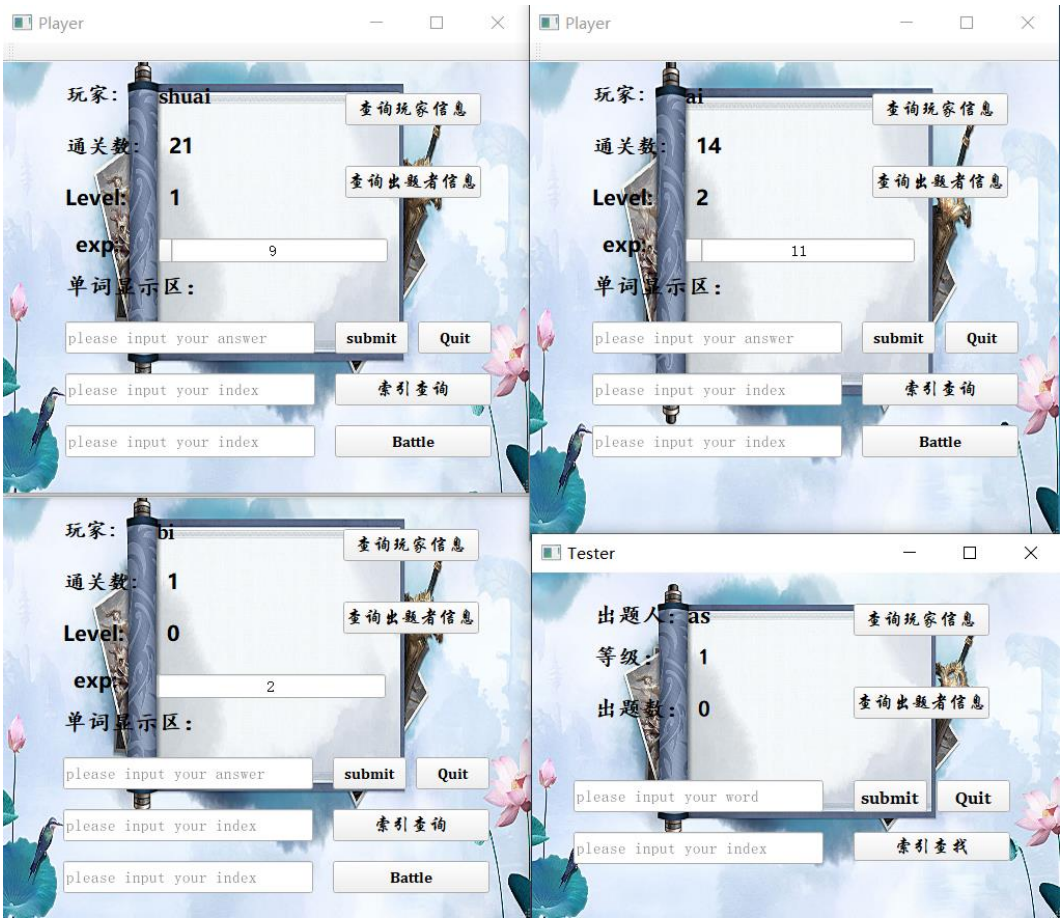


图 1

Player Information

	玩家	等级	经验值	已经闯关数	is_land
1	ai	2	11	14	1
2	bi	0	2	1	1
3	bupt	2	14	8	0
4	gust	0	0	0	0
5	mircale	0	0	0	0
6	peng	2	10	6	0
7	putty	0	0	1	0
8	shuai	1	9	21	1

Tester Information

	出题者	等级	已经出题数	is_land
1	as	1	0	1
2	bupt_peng	0	0	0
3	pengs	1	3	0
4	pengsh	0	1	0
5	shuai	4	1	0

图 0

六、实验总结

（一） 易错点

在处理从服务器方接受信息的易错之处就是：在 socket 中读取信息可能需要时间，因此我需要让程序给留出一点这个时间让客户端将信息读取出来并赋值给相应的变量，这就需要下面的函数来支持：

```
//让它停在这里 0.5s(程序还在进行)
QEventLoop eventloop1;
QTimer::singleShot(500,&eventloop1,SLOT(quit()));
eventloop1.exec(0);
```

当然这个也可以通过前面我说的信号槽机制加以解决

（二） 优点

整个实验的代码风格全部按照老师上课所讲的要求实现，易读性较高；

界面对用户比较友好，用户易于理解；

通过多线程实现对多用户并发计算；

还有其他的一些细节：比如说玩家进行游戏时，每答出一提，相应的玩家信息就会通过服务器在文件中更新，而不是整个游戏结束之后在更新，这样做就会让那些同时在线的玩家能够看到**实时玩家信息情况并且得到实时排名信息。**

（三） 收获

通过这次程序设计实践，复习了很多关于 C++方面的知识：类的封装、类的继承、虚函数、重载等等。更加理解了“**面向对象**”的含义：通过对某一对象进行操作，达到所需效果这样的思想。这对我们进行软件研发很关键。程序和信息之间通过这个对象来连接，程序需要相关信息是，会从文件或者数据库或者那个调出信息并将这些信息赋值给这个对象，然后我们再对这个对象进行具体的操作，这样的思想使得我们在编写程序、设计软件的过程中有迹可循，以更加熟悉和普遍的思维模式，更高效地编写出更具有可维护性、可扩展性、可复用性的程序软件。

另外通过这次实践深刻理解了 Client-Server 模式的通信以及多线程模式，客户端发一个请求，服务器作出相应的相应，两者之间通过套接字 socket 进行通信。

（四）改进

对于双人对战系统，有时候可能出现 bug，这个对战有点复杂，我这里实现

的只是简单的模拟一下大致效果，并没有实际对战系统的优化，可能需要更加细分通信格式吧。

七、实验源码

（见附件）