单词记忆游戏实验报告

2017211305 班 2017211240 于海鑫

版本：ϵ

更新：*May 14, 2019*

本文档为“面向对象程序设计实践（C++）”课程综合实验“单词消除游戏系统设计与开发”的实验报告。

1 总体介绍

此次实验实现了跨平台的单词消除游戏系统，并可以通过互联网进行多人联机对战。该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。一关由一轮或者多轮组成。实验要求我们按照如下三步实现整个系统：

(1). 实现最基本的游戏功能
(2). 增加游戏的可玩性，并设置升级策略
(3). 将游戏从单机拓展为服务器多人游戏平台

除去最基本的要求外，此次实验还通过 QT 实现了可以跨平台的图形界面。本系统使用了面向对象的思想进行设计，以下将分版本详细介绍各个模块。

2 第一版

2.1 概览

整个程序围绕一个最基础的 **BaseUser** 类进行开发。后续的 **GamerHandler** 以及 **AdminHandler** 都是对此类进行派生得到的。如下是该类的定义：

```
struct BaseUser {
    QString userName;
    QString realName;
    int levelPassed;
    int experience;
};
```

这个类主要是为了与数据库交互，数据库中的表的格式与之完全一致，以下是创建用户库的 SQL 语句：

```
CREATE TABLE IF NOT EXISTS user (
    utype INT NOT NULL,
    uname TEXT PRIMARY KEY NOT NULL,
    rname TEXT NOT NULL,
    level INT NOT NULL,
    exp INT NOT NULL,
    password TEXT NOT NULL
);
```

为了简化逻辑，我们的的数据库内仅存经验，等级通过再图形界面内实时生成。其对应关系为：

$$level = \frac{\sqrt{1 + \frac{8 \times experience}{50}} + 1}{2} + 1$$

2.2 图形界面

为了实现较好的跨平台体验，我们的图形界面使用 QT Quick 进行实现。用于实现图形界面的文件全部位于 **ui** 文件夹下。7 个 QML 文件对应于我们实现的 6 张页面。以下是详细信息。

- **ui/MemoryGame.qml** 定义窗口，初始化界面
- **ui/LoginPage.qml** 登录界面
- **ui/RegisterPage.qml** 注册界面

- **ui/GamePage.qml** 游戏主界面
- **ui/AdminPage.qml** 管理员界面，可以在此添加词汇
- **ui/InfoPage.qml** 展示当前用户的信息
- **ui/UsersPage.qml** 展示当前全部用户的信息

2.3 控制模块

这一部分主要是处理用户的输入，并将用户的输入反映到相应的数据库内。这一部分的类命名为 ***Handler**，这一部分类会继承 **BaseUser** 类以及 **QObject** 类，并通过 QT 提供的机制暴露到图形界面内。图形界面内用户的输入会被转换为对应 **Handler** 类的函数调用。**Handler** 类则通过封装好的数据库类修改数据库内的数据。例如，作为 **GamerHandler** 以及 **AdminHandler** 的基类的 **UserHandler** 定义如下：

```
class UserHandler : public QObject, BaseUser {
    Q_OBJECT

    Q_PROPERTY(QString userName READ getUserName WRITE setUserName
        NOTIFY
    userNameChanged)
    Q_PROPERTY(QString realName READ getRealName WRITE setRealName
        NOTIFY
    realNameChanged)
    Q_PROPERTY(int levelPassed READ getLevelPassed WRITE
        setLevelPassed NOTIFY
    levelPassedChanged)
    Q_PROPERTY(int experience READ getExperience WRITE setExperience
        NOTIFY
    experienceChanged)

public:
    explicit UserHandler(QObject *parent = nullptr);
    QString getUserName() const;
    void setUserName(QString userName);
    QString getRealName() const;
    void setRealName(QString realName);
    int getLevelPassed() const;
    void setLevelPassed(int levelPassed);
    int getExperience() const;
    void setExperience(int experience);
```

```
23      Q_INVOKABLE void updateUser();
24
25  signals:
26      void userNameChanged();
27      void realNameChanged();
28      void levelPassedChanged();
29      void experienceChanged();
30      };
```

其作用就是为图形界面提供信息，并通过 **updateUser** 函数更新图形界面。其子类中方法的实现方式与之类似。

2.4 数据库连接

这一部分主要是处理与数据库连接相关的内容，主要是将对应的 SQL 语句包装为函数。这一部分的类命名为 ***DB**。此处的类只可以有一个实例，故理所当然的使用单例模式进行实现。我们可以使用 ***DB::Instance()** 来获得取唯一的访问点。

除去上文介绍的类以外，我们的第一版程序还使用 C++ 自定义实现了 **QTableView** 的 Model，以此来实现视觉效果更好的图表。

3 第二版

第二版并无过大的更新，主要是在图形界面中实现多轮游戏的逻辑。并在添加单词时自动为其分配难度。核心更改如下：

```
1  int getDifficulty(QString s) {
2      if (s.length() >= 7) {
3          return 3;
4      } else if (s.length() >= 4) {
5          return 2;
6      }
7      return 1;
8  }
```

图形界面的更在在此不表。

4 第三版

这一版主要是实现多人连接。在这里我通过使用 C++ 标准库自带的 thread 实现了较为简单的网络通信框架。全部实现在 **socket/** 文件夹内。

4.1 服务器部分

服务器部分由前两版中的数据库连接模块加上收发数据的函数构成。需要注意的是 Qt 默认不支持跨线程的数据库查询 (Qt 5.12 引入的 **FEATURE**)，需要自己对 Qt 的源码进行调整使之适应我们的服务器代码。源码的修改很简单，不再赘述。

4.2 客户端部分

这一部分是由前两版中的图形界面以及控制模块加上客户端的数据处理代码构成的。为了简化程序逻辑，整个程序基于 Qt 的信号和槽机制开发。事实证明我们实现的通信框架和 Qt 内置的信号和槽机制相性极佳。在客户端中，数据处理模块也按照单点模式进行实现。数据处理的主要任务就是通过信号告知之前的各个模块数据的改变。部分代码如下：

```
1  static void onLoginResult(__attribute__((unused)) SocketClient *
       client, std::string result) {
2      QStringList resultList = QString::fromStdString(result).split('$'
           );
3      Client::Instance().emitLoginResult(
4          resultList.at(0).toInt() == 0 ? false : true, resultList.at
               (1),
5          resultList.at(2), resultList.at(3).toInt(), resultList.at(4).
               toInt());
6  }
7
8  static void onUid(__attribute__((unused)) SocketClient *client, std::
       string result) {
9      Client::Instance().setTag(QString::fromStdString(result).toUInt()
           );
10 }
```

在接收到类似的请求时，函数被执行，部分结果通过信号送入图形界面，界面就会根据

信号进行相应的更改。服务器中的逻辑与之类似。

5 额外工作

此次实验额外实现了查看是否在线的功能，其实现原理如下：服务器端存在一个集合，用以保存已经登录的账号的用户名，当需要查看某个用户是否在线时只需查表即可。同时该集合还用于判断是否有重复登录。

除此之外，在服务器端清空 **SocketClient*** 组时，我实现了一个较为简单的退出时执行 (类似 **Go** 语言的 **defer**，但是执行顺序与之相反)，代码如下：

```
#define DEFER_1(x, y) x##y
#define DEFER_2(x, y) DEFER_1(x, y)
#define DEFER_0(x) DEFER_2(x, __COUNTER__)
#define defer(expr) auto DEFER_0(_defered_option) = deferer([&]() {
    expr; })

template <typename Function> struct doDefer {
    Function f;
    doDefer(Function f) : f(f) {}
    ~doDefer() { f(); }
};

template <typename Function> doDefer<Function> deferer(Function F) {
    return doDefer<Function>(F);
}
```

这段代码很好地表现出了 C++ 中的模板与 C 中的宏结合后的灵活性。

A 附录：静态编译 Qt 的 PowerShell 脚本

该脚本为去年计算导论与程序设计实践课程中根据开源代码修改的来。地址如下：

windows-build-qt-static.ps1