# 第七周实训报告

这周主要把之前遗漏的排行榜工作完成了，并且对部分代码进行了重构，提高了游戏运行时的效率。

到目前为止，游戏的基本工作已经结束了。

原本计划的音乐播放器和联网功能目前看实现不了，可能是由于Qt的版本问题，一直加入不了模块。

## 排行榜的实现

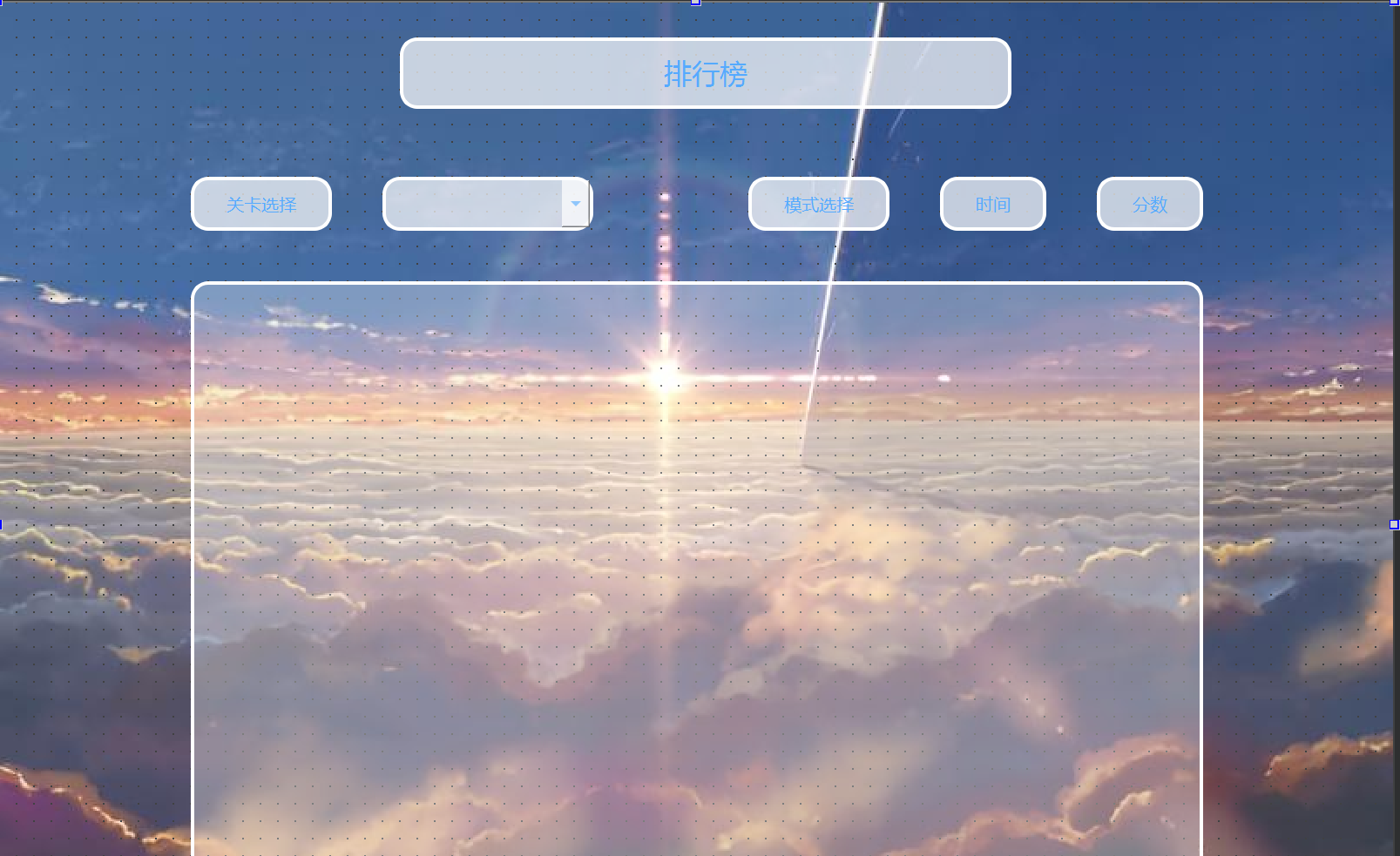
排行榜主要实现的功能如下：

1.能够对玩家的游戏记录进行排名，主要有两个游戏标准：时间和分数

2.对此会生成两个完全不同的排行榜：时间排行榜和分数排行榜

3.时间排行榜只是根据玩家的游戏时间进行排名，而分数排行榜会根据玩家的操作步数和撤回步数进行排名

采用的主要是QTablevView和QStandardItemModel和QCombox实现



玩家通过点击QCombox，选择关卡号，然后点击“时间”或者“分数”按钮,，就可以生成对应关卡的排行榜。

#ifndef RANKING\_H
  
#define RANKING\_H
  
  
#include <QWidget>
  
#include <QListView>
  
#include <QStandardItem>
  
#include <QSortFilterProxyModel>
  
#include <QFile>
  
#include "Player.h"
  
namespace Ui {
  
class Ranking;
  
}
  
  
class Ranking : public QWidget
  
{
  
 Q\_OBJECT
  
  
public:
  
 explicit Ranking(QWidget \*parent = nullptr,Player \*onlinePlayer=nullptr);
  
 ~Ranking();
  
 void loadData(const int level); //负责从文件导入原始数据，生成排行榜
  
 void updateData(const int level); //负责对文件数据进行更新，并且导入文件
  
 void updateRanking(const int levelNeedUpdate); //负责对排行榜的内容进行实时更新
  
 void updateTimeModel(); //对时间排行榜进行更新，并且显示
  
 void updateScoreModel(); //对分数排行榜进行更新，并且显示
  
 struct myData //封装的一个小用户信息类，用于接受游戏记录
  
 {
  
 QString m\_name; //用户名
  
 int time; //时间
  
 int score; //分数
  
 };
  
private:
  
 Ui::Ranking \*ui;
  
 QStandardItemModel \*time\_model; //时间model
  
 QStandardItemModel \*score\_model; //分数model
  
 Player \*m\_player; //用户类指针，用来指向当前游玩的用户
  
 std::vector<myData> playerData[13]; //vector，用于接收12个文件关卡的游戏数据
  
};
  
  
  
#endif // RANKING\_H

Ranking的构造函数

Ranking::Ranking(QWidget \*parent,Player \*onlinePlayer)
  
 : QWidget(parent)
  
 , ui(new Ui::Ranking),m\_player(onlinePlayer)
  
{
  
 ui->setupUi(this);
  
  
 time\_model=new QStandardItemModel(this);
  
 score\_model=new QStandardItemModel(this);
  
  
 //默认1开始，下载十二个关卡的数据
  
 for(int i=1;i<=12;i++)
  
 {
  
 loadData(i);
  
 }
  
  
 //多个地图选项布局，对combox进行初始化
  
 QList<QString> com\_list;
  
 for(int i=1;i<=12;i++)
  
 {
  
 com\_list.append(QString("%1").arg(i));
  
 }
  
 ui->chooseMap->addItems(com\_list);
  
 ui->chooseMap->setCurrentIndex(0);
  
  
  
 ui->view->setModel(time\_model); //默认设置当前model为time\_model
  
 connect(ui->timeBtn,&QPushButton::clicked,this,&Ranking::updateTimeModel); //槽函数，点击时间按钮会生成时间排行榜
  
 connect(ui->scoreBtn,&QPushButton::clicked,this,&Ranking::updateScoreModel); //槽函数，点击分数按钮会生成分数排行榜
  
  
 ui->view->setHorizontalScrollBarPolicy(Qt::ScrollBarAlwaysOff); //设置水平滚动条隐藏
  
 ui->view->setVerticalScrollBarPolicy(Qt::ScrollBarAlwaysOff); //设置竖直滚动条隐藏
  
 ui->view->setEditTriggers(QAbstractItemView::NoEditTriggers); //设置为无法编辑，防止修改数据
  
 ui->view->horizontalHeader()->setSectionResizeMode(QHeaderView::Stretch); //设置水平方向铺满排行榜框
  
  
 //设置背景
  
 QImage background(":/images/resource/picture/背景.jpg");
  
 background=background.scaled(this->width(),this->height(),Qt::IgnoreAspectRatio);
  
 QPalette palette;
  
 palette.setBrush(this->backgroundRole(), QBrush(background));
  
 this->setPalette(palette);
  
}

updateTimeModel函数（updateScoreModel函数相似，不再展示）

void Ranking::updateTimeModel()
  
{
  
 QString currentIndex=ui->chooseMap->currentText(); //获取当前combox的页面编号
  
 bool ok=false;
  
 qDebug()<<"showRanking"<<currentIndex;
  
 int level=currentIndex.toInt(&ok);
  
 qDebug()<<"排行榜更新成功"<<ok;
  
 if(level==0||m\_player->getUserName()=="tourist") //如果是游客模式，则不能查看排行榜，或者当前获取需要更新的关卡编号为0，则返回
  
 {
  
 return;
  
 }
  
 std::sort(playerData[level].begin(),playerData[level].end(), //按照时间进行排序
  
 [](const myData &a,const myData &b)
  
 {
  
 return a.time<b.time;});
  
  
 time\_model->clear(); //对原来的model清空
  
 time\_model->setHorizontalHeaderLabels(QStringList()<<"用户名"<<"时间");
  
 int cnt=0;
  
 for(const auto &item:playerData[level]) //重新填充model的内容
  
 {
  
 QStandardItem \*name\_item=new QStandardItem(QString(item.m\_name)); //用户名
  
 QStandardItem \*time\_item=new QStandardItem(QString::number(item.time)); //游戏时间
  
 name\_item->setTextAlignment(Qt::AlignHCenter);
  
 name\_item->setSizeHint(QSize(100,50));
  
 time\_item->setTextAlignment(Qt::AlignHCenter);
  
 time\_item->setSizeHint(QSize(100,50));
  
 time\_model->setItem(cnt,0,name\_item); //设置坐标
  
 time\_model->setItem(cnt,1,time\_item); //设置坐标
  
 cnt++;
  
 }
  
 emit time\_model->dataChanged(time\_model->index(0, 0), time\_model->index(cnt-1, 1)); //发送信号，对time\_model进行更新
  
 ui->view->setModel(time\_model); //使用新的time\_model
  
 ui->view->horizontalHeader()->setSectionResizeMode(QHeaderView::Stretch); //设置水平方向铺满排行榜框
  
}

loadData函数与updateData函数

void Ranking::loadData(const int level)
  
{
  
 //下载文件数据
  
 QString fileName=QString("./time%1.bin").arg(level);
  
 QFile file(fileName);
  
 QDataStream in(&file);
  
  
 if(!file.open(QIODevice::ReadOnly))
  
 {
  
 qDebug()<<"错误 无法打开文件进行读取 并生成排行榜";
  
 return;
  
 }
  
 qDebug()<<"排行榜文件夹打开成功";
  
 while(!in.atEnd())
  
 {
  
 myData temp;
  
 in>>temp.m\_name>>temp.time>>temp.score;
  
 playerData[level].push\_back(temp);
  
 }
  
 file.close();
  
  
 for(auto &item:playerData[level])
  
 {
  
 qDebug()<<level<<"关卡生成的数据"<<item.m\_name<<item.time<<item.score;
  
 }
  
}
  
  
void Ranking::updateData(const int levelNeedUpdate)
  
{
  
 //更新文件数据
  
 int level=levelNeedUpdate;
  
 if(level==0||m\_player->getUserName()=="tourist")
  
 {
  
 return;
  
 }
  
 QString fileName=QString("./time%1.bin").arg(level);
  
 QFile file(fileName);
  
 QDataStream in(&file);
  
  
 if(!file.open(QIODevice::WriteOnly))
  
 {
  
 qDebug()<<"错误 无法打开文件 对排行榜进行修改";
  
 return;
  
 }
  
 qDebug()<<"打开文件成功 并且开始修改";
  
 for(const auto &item: playerData[level])
  
 {
  
 in<<item.m\_name<<item.time<<item.score;
  
 }
  
 file.close();
  
}

updateRanking函数

void Ranking::updateRanking(const int levelNeedUpdate)
  
{
  
 //更新playerData
  
 int level=levelNeedUpdate;
  
 myData temp; //设置将要更新的用户信息
  
 temp.m\_name=m\_player->getUserName();
  
 temp.time=m\_player->getTime(level);
  
 temp.score=500-m\_player->getSteps(level)-m\_player->getBackSteps(level)\*10;
  
  
 bool checkOK=false; //判断是否存在
  
 qDebug()<<"before update level"<<level<<"size: "<<playerData[level].size();
  
 for(int i=0;i<playerData[level].size();i++)
  
 {
  
 if(temp.m\_name==playerData[level][i].m\_name)
  
 {
  
 if(temp.time<playerData[level][i].time) //如果当前的游戏记录比之前的更好
  
 {
  
 playerData[level][i].time=temp.time; //更新游戏记录
  
 checkOK=true; //标记为：已经存在
  
 }
  
 else if(temp.score>playerData[level][i].score)
  
 {
  
 playerData[level][i].score=temp.score;
  
 checkOK=true;
  
 }
  
 else
  
 {
  
 checkOK=true; //标记为：已经存在
  
 }
  
 }
  
 if(checkOK) break;
  
 }
  
  
 if(!checkOK) playerData[level].push\_back(temp); //没有检查到，说明是新用户
  
  
 qDebug()<<"排行榜更新成功";
  
 qDebug()<<"after update level"<<level<<"size: "<<playerData[level].size();
  
}

目前已经进入了收尾工作，计划对原有的代码进行重构，在生成某些地图时有些小卡顿或者不尽人意的地方，打算进行重写。