作业报告

一、程序功能介绍

此程序为一款 4Key 音乐游戏，有开始界面、选曲界面、设置界面、游戏界面和结算界面。从开始界面可以进入选曲界面，在选曲界面可以进行选曲，选择相应的曲目并点击开始按键之后可以开始游戏，从选曲界面也可以进入设置界面调整谱面流速和偏移。在游戏界面中，玩家可以通过键盘上 Q, W, O, P 四个按键控制相应的轨道，在音符落到判定线上时按下相应轨道对应的按键根据判定得到分数，最终在游戏结束后进入结算界面显示各种信息，之后可以回到选曲界面继续游戏。此外，这个程序也支持在现有的曲目和谱面之上导入其他的符合格式的曲目和谱面。

二、项目各模块与类设计细节

本项目主要按照界面来分各个模块，接下来就按照开始界面、选曲界面、设置界面、游戏界面和结算界面这几个模块依次介绍。此外，还有两个类由各个界面通用，因此先进行介绍。

1.通用模块

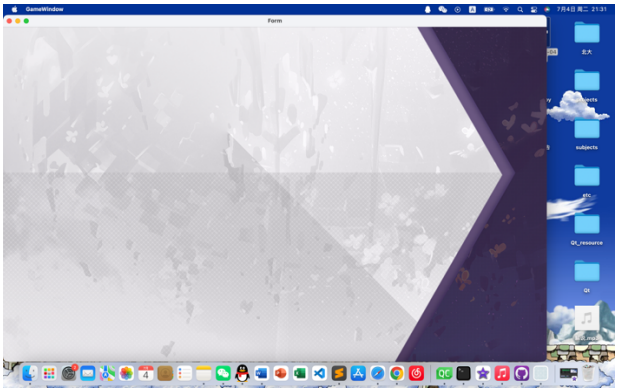
涉及到的类有 OpenAnimation 类和 ShutterAnimation 类，主要功能是在页面切换时播放切入和切出的动画。两个类较为相像，唯一的差别就是两个 QLabel 对象的运动方向相反，因此下文以介绍 OpenAnimation 类为例。

```
class OpenAnimation : public QObject
{
    Q_OBJECT
public:
    explicit OpenAnimation(QWidget *parent = nullptr,int duration=1000,int windowWidth=1280,int windowHeight=800);
    QWidget* parentWindow;
    QPropertyAnimation* animation[2]; //0-左侧 1-右侧
    QLabel* shutter[2]; //0-左侧 1-右侧
    QAudioOutput* audioOutput;
    QMediaPlayer* player;
    ~OpenAnimation();
signals:
    void finished();
public slots:
    void play();
    void emitFinished();
    void hide();
};
```

上图为 OpenAnimation 类的定义。

parentWindow 保存其所属的 QWidget 的指针，以便于在窗体内生成 QLabel 对象。animation 保存的是两个 QPropertyAnimation 的对象的指针，shutter 保存的是两个 QLabel 对象的指针。这几个变量主要实现了两张图片从闭合到打开的动画，具体实现方法为将两个 QLabel 对象的背景图设为两张可以嵌合并完全覆盖窗口的图片，之后再利用 QPropertyAnimation 的相关设置使其逐渐水平地从两向移出窗口（其中有个小技巧是使用缓动曲线，也就是 EasingCurve 使得动画的过程更丝滑）。

```
//动画设置
for(int i=0;i<2;i++){
    animation[i]=new QPropertyAnimation(this);
    animation[i]->setEasingCurve(QEasingCurve::InQuad);
    animation[i]->setPropertyName("pos");
    animation[i]->setTargetObject(shutter[i]);
    animation[i]->setStartValue(shutter[i]->pos());
    if(i==0) animation[i]->setEndValue(QPoint(-shutterSize[i][0],windowHeight/2-shutterSize[i][1]/2));
    else animation[i]->setEndValue(QPoint(windowWidth,windowHeight/2-shutterSize[i][1]/2));
    animation[i]->setDuration(duration);
}
```



上图为闭合时的窗口截图。

audioOutput 和 player 主要用于播放动画音效。

```
//音效设置
player=new QMediaPlayer(parent);
audioOutput=new QAudioOutput(parent);
player->setAudioOutput(audioOutput);
player->setSource(QUrl(QString::fromStdString(PATH+"soundeffects/shutter_open.wav")));
```

OpenAnimation 类的构造函数中除了父窗体的指针之外还有 duration（对应动画的时长），windowLength 和 windowHeight（对应父窗体的尺寸）。这个类对外提供了 play()（开始动画）的槽和 finished()（标志动画结束）的信号。

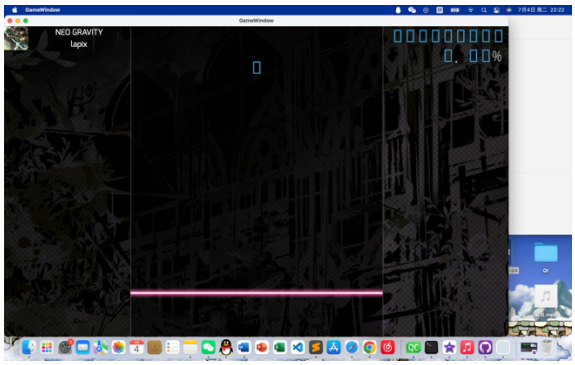
2.游戏界面

与之相关的主要有 Note 类，NoteHandler 类和 GameWindow 类。

其中 Note 类主要用于管理一个单独的音符实体，NoteHandler 类用于管理整个窗口中的所有音符及其判定，还有一些数据的处理（分数，准确率等），GameWindow 类中主要进行的是整个界面的设计（调整各个控件的位置）。

（1）GameWindow 类

窗体控件的各种显示（比如这个窗口中的各个文字，背景之类的设置）比较繁杂并且实际上就是不断地 resize, move, 设置字体，设置 stylesheet 等等之类的反复工作，因此省略，大体效果如下图。中间一部分有判定线和下部的衬底，中上方是连击数和判定（图中没有，因为还没有音符被判定）的显示，左上方是封面图以及曲目基本信息，右上方是分数和准确度。



在整个项目的设计中，有很多技巧是类似的，因此在此就借 GameWindow 类介绍一下（不过由于这个窗口中没有按钮，所以留与后续介绍）。

1）切换动画

使用 OpenAnimation 类和 ShutterAnimation 类实现。
OpenAnimation 类的 play()在 GameWindow 类的构造函数中被调用，在初始化 GameWindow 界面之后便开始动画。
ShutterAnimation 类在最后一个音符落下一段时间之后再进行，因此使用了 QTimer::singleShot 进行计时，并在一定时间之后再创建新窗口。

```
//结束后切换至结算界面
resultShutterAnimation=new ShutterAnimation(gameWindow);
int shutterStartTime=BASETIME+lastNoteTime+TAILTIME;
QTimer::singleShot(shutterStartTime,Qt::PreciseTimer,resultShutterAnimation,SLOT(play()));
```

2）新窗口的创建和原窗口的关闭
窗口的管理主要通过 QWidget* currentWindow 这个全局变量进行，每当窗口进行切换的时候就会关闭当前窗口，并且 currentWindow＝new （目标窗口的类的名字）（参数表），然后 currentWindow -> show()完成窗口的切换，例如下图。

```
//关闭窗口和开启结算界面
QTimer::singleShot(shutterStartTime+2000,Qt::PreciseTimer,nullptr,[=]{
    currentWindow=new ResultWindow();
    currentWindow->show();
});
QTimer::singleShot(shutterStartTime+2000,Qt::PreciseTimer,gameWindow,SLOT(close()));
```

在这个过程中不存在多窗口切换的时候指向原有的窗口的指针被抛弃导致内存泄漏的问题，因为在每个窗口类的构造函数中都加上了如下图的内容：

```
this->setAttribute(Qt::WA_DeleteOnClose);
```

其含义是将窗口设置为关闭即删除的状态，所以无需顾虑删除的事情。

（2）Note 类
以下为 Note 类的定义。

```
class Note:public QObject{
    Q_OBJECT
public:
    int lane; //轨道 1,2,3,4
    int moveTime; //应该移动时的毫秒数
    QLabel* note; //实体
    QPropertyAnimation note_drop; //移动动画
    bool judged;
    int judgement;

    Note(QWidget* gameWindow,int duration,int _moveTime,int _lane);
    void reset(int _lane,int _moveTime);
    ~Note();

public slots:
    void startMove();
    void passData();

signals:
    void dataUpdate(bool _judged,int _judgement,int _lane);
};
```

其中最主要的是 QLabel* note，其背景图被换为音符的图片，也就是说这个 QLabel*所管理的就是实际上显示的对象。note_drop 通过一个改变 note 的 Y 坐标的动画实现音符“下落”的动画效果。startMove()这个槽的效果是使音符实体开始下落，与定时器连接（在 NoteHandler

类中进行）使得音符可以在正确的时间下落。

（3）NoteHandler 类

NoteHandler 类是用于管理音符的类，同时也是游戏界面中最核心的类，其定义如下。

```
class NoteHandler:public QObject{ //完成startHandle
    Q_OBJECT
public:
    QWidget* gameWindow;
    int totalNotes;
    int moveTimeArr[10000];
    int laneArr[10000];
    std::queue<Note*> handledNotes[5];
    std::stack<Note*> notePool;
    QMediaPlayer* player;
    QAudioOutput* audioOutput;
    OpenAnimation* openAnimation;
    ShutterAnimation* resultShutterAnimation;
    NoteHandler(QWidget* _gameWindow);
    ~NoteHandler();
    //统计
    int total_perfect;
    int total_great;
    int total_good;
    int total_miss;
    int combo;
    int max_combo;
    double accuracy;
    int score;
    int curComboMul;
    int comboMul[4]; //*0.01
    //显示从低位到高位
    QLabel* scoreEntity[9];
    QLabel* accuracyEntity[7];
    QLabel* comboEntity[6];
    int comboDigitToPos[7][2];
    QLabel* judgementEntity;
    int inProgressJudgementShow;
    QPixmap background[17]; //0-9:数字0-9 10:点 11:百分号 12:MISS 13:GOOD 14:GREAT 15:PERFECT 16:空白

public slots:
    void judge(int lane);
    void dataUpdate(bool judged,int judgement,int lane);
    void dataShow(int judgement);
    void judgementShow();
    void playMusic();
    void outputResult();

signals:
    void dataShowSig(int judgement); //0-MISS 1-GOOD 2-GREAT 3-PERFECT
};
```

其主要功能是根据 std::string PATH（文件所在位置）和 std::string songName（曲名，由选曲界面设置）这两个全局变量读取谱面文件和设定文件，从而得到音符应当落下的时机，然后用 QTimer::singleShot 设置好对应音符的下落时间。此外，NoteHandler 类还负责了音符的判定，数据的显示以及音乐的播放。

其中音乐的播放的方式和 OpenAnimation 中相同，不多赘述。

数据的显示无非是根据相应的判定结果和公式计算来得到相应的数字并且通过改变 QPixmap 来显示对应的结果，因此也不多说。唯一需要说明的是分数的计算公式：每一个判定（PERFECT，GREAT，GOOD，MISS）有其对应的基础分数（在这个项目中分别是 1000，700，400，0），得到相应的分数会直接加上，但与此同时也会有附加分数，根据过去所获得的判定所决定。最开始的附加比例是 0，得到 PERFECT 判定这个比例加 0.01，得到后三种则分别减 0.02，0.08 和 0.32，这个比例在动态变化的过程中不会小于 0，也不会大于 1。随后附加分数便是基础分数乘以附加比例。

接下来来解释 NoteHandler 类中最重要的部分——判定。

其大致实现是这样的：首先，根据判定线的位置、音符的流速以及各个判定区间的毫秒数计算各个判定所对应的音符位置，比如按下对应的按键的时候音符在离判定线足够近的位置的话，那就是 PERFECT 判定，稍微远一点的话就是 GREAT，再远一点的话就是 GOOD，如果没有落到判定区域内的话就什么都不做，而如果掉到了判定区域之下的话就直接判为 MISS，这个过程就是在计算各个区间的位置。

```
//计算判定范围
double PIXEL_PER_SEC=((double)windowHeight)/((double)DURATION);
UPPER_GOOD=laneLength-((int)(((double)GOOD)*PIXEL_PER_SEC));
LOWER_GOOD=laneLength+((int)(((double)GOOD)*PIXEL_PER_SEC));
UPPER_GREAT=laneLength-((int)(((double)GREAT)*PIXEL_PER_SEC));
LOWER_GREAT=laneLength+((int)(((double)GREAT)*PIXEL_PER_SEC));
UPPER_PERFECT=laneLength-((int)(((double)PERFECT)*PIXEL_PER_SEC));
LOWER_PERFECT=laneLength+((int)(((double)PERFECT)*PIXEL_PER_SEC));
```

大致的判定过程已经在上面一段说完了，但是关于 MISS 需要考虑两种情况，一种是完完全全地漏了，音符一直落到窗体底部而这期间玩家并没有按下按键，另一种是音符没有完全落到底部但是已经脱离了判定区域的时候玩家按下了按键。这两点的区别是由于 NoteHandler 管理音符的方式，NoteHandler 中所有待判定的音符都储存在一个个的队列中，每一个队列对应一个轨道，因此需要先判定完第一个音符并且将其从队列取出之后才能进行下一个音符的判定，对于第一种情况，对应的 Note 的 note_drop 会释放 finished()信号，会使 NoteHandler 进行相应的处理，不会妨碍下一个音符的判定。但是第二种 MISS 会导致一种问题：可能下一个音符已经进入了判定区域，但是第一个音符却仍然在下落，这时我们不可以只判定第一个音符而不判定下一个音符，因为第一个音符已经在判定范围之外了，但是这样的话第一个音符又会占着队列的首位而导致我们想判定的音符无法被判定。因此，在这里的解决办法就是如果遇到这种情况，提早判定这个音符并将其移出队列，然后同时对第二个音符进行判定，在 NoteHandler::judge()中的体现如下。

```
if(y>LOWER_GOOD){
    if(!handledNotes[lane].empty()){
        qDebug()<<"PREJUDGED";
        judgedNote->judged=1;
        judgedNote->judgement=0;
        dataUpdate(1,0,lane);
        judgedNote=handledNotes[lane].front();
        y=judgedNote->note->pos().y();
    }else{
        return;
    }
}
```

一个被正常判定（PERFECT、GREAT、MISS）的音符的信号传递过程是这样的：对应键盘按键被按下 -> GameWindow::keyPressed() -> NoteHandler::judge() -> NoteHandler::dataUpdate() -> 数据更新；而对于第一种情况的 MISS 的音符而言，信号传递过程是：QPropertyAnimation::finished() -> Note::passData() -> Note::dataUpdate() -> NoteHandler::dataUpdate -> 数据更新；对于第二种情况的 MISS 的音符而言，信号传递的过程是：对应键盘按键被按下-> GameWindow::keyPressed() -> NoteHandler::judge() -> NoteHandler::dataUpdate() -> 数据更新，但是过程中也会判定下一个音符，并且阻断这个音符的 Note::passData() -> Note::dataUpdate()的通路，避免其被反复判定。

此外，在游戏结束后 NoteHandler 类会统计信息输出到特定的 txt 文件中，以便于控制结算界面的类读取并显示。

3.开始界面

涉及到 StartWindow 类，定义如下。

```
class StartWindow : public QWidget
{
    Q_OBJECT

public:
    explicit StartWindow(QWidget *parent = nullptr,bool firstOpen=false);
    ~StartWindow();
    QLabel* definitelynot;
    QLabel* arcaea;
    QPushButton* play;
    QPushButton* quit;
    QLabel* quitContent;
    OpenAnimation* openAnimation;
    ShutterAnimation* shutterAnimation;

public slots:
    void goToSelection();

private:
    Ui::StartWindow *ui;
};
```

（1）利用 QPalette 和 QPixmap 将相应资源文件作为背景；
（2）将标题置于界面中上部；
（3）设置 play 按钮，用 setStyleSheet 改变按钮的背景，并且利用 connect 函数将按钮被 press 时的背景改为另一背景文件，实现 press 过程中按钮高亮的效果，更加真实立体，用界面指针实现到选歌界面的变换（其他的所有带有按下/松手后又对应效果的按钮用的是同样的方法）；
（4）设置 quit 按钮，实现按下时退出项目的功能。同样有按下时的高亮效果。

4.选曲界面
涉及到 SelectionWindow 类，定义如下。

```
class SelectionWindow : public QWidget
{
    Q_OBJECT

public:
    int current_first_song_index;
    int total_songs;
    QLabel* covers[5];
    QLabel* metadata[3]; //0-曲名 1-by 2-曲师
    QLabel* backContent;
    QPushButton* start_game;
    QPushButton* settings;
    QPushButton* up;
    QPushButton* down;
    QPushButton* back;
    std::string songnames[100];
    OpenAnimation* openAnimation;
    ShutterAnimation* gameShutterAnimation;
    ShutterAnimation* settingsShutterAnimation;
    ShutterAnimation* backShutterAnimation;
    explicit SelectionWindow(QWidget *parent = nullptr,int currentSong=0);
    ~SelectionWindow();

public slots:
    void go_up();
    void go_down();
    void setMetadata(int ind);
    void startGame();
    void goToSettings();
    void goToStart();

private:
    Ui::SelectionWindow *ui;
};
```

核心功能：实现曲目的选取；
实现方法：
（1）从 song.txt 文件中读取曲名，存为 songnames；
（2）初始化按钮组件 QPushButton* up、down，实现曲目的切换；
（3）加载曲目信息和每个曲目的封面图；
（4）初始化按钮组建 QPushButton* start、settings、back，分别实现开始游戏、进入设置界

面、返回开始界面；
（5）函数 go_up、go_down 实现改变 current_first_song_index，从而改变了所选中的曲目，进而将选中歌曲显示在中央；
（6）函数 setMetadata()实现曲目封面的变换；
（7）函数 startGame() , goToSettings() , goToStart()分别实现开始游戏、进入设置界面、进入开始界面；
（8）界面之间的切换动画通过 OpenAnimation 与 ShutterAnimation 类实现。

5.设置界面
涉及到 SettingsWindow 类，定义如下。

```
class SettingsWindow : public QWidget
{
    Q_OBJECT

public:
    explicit SettingsWindow(QWidget *parent = nullptr);
    ~SettingsWindow();
    QLabel* mask;
    QLabel* settingsHead;
    QLabel* settingsWindow;
    QLabel* headTitle;
    QLabel* speedTitle;
    QLabel* offsetTitle;
    QSpinBox* speed;
    QSpinBox* offset;
    QPushButton* back;
    QLabel* backContent;
    OpenAnimation* openAnimation;
    ShutterAnimation* shutterAnimation;

public slots:
    void goToSelection();

private:
    Ui::SettingsWindow *ui;
};
```

核心功能：调整谱面流速（Speed）与偏移（Offset）。从而能够匹配玩家的游戏喜好，并且能够适配不同的设备。
实现方法：
（1）初始化外观组件；
（2）初始化按钮组件 QPushButton back，实现返回选曲界面的功能；
（3）函数 goToSelection()实现返回选曲界面的功能；
（4）初始化数值设定模块 QSpinBox* speed 和 offset，实现点击修改与键盘输入修改的功能；
（5）其中谱面流速和偏移的储存是通过利用 fstream 读取/写入特定的 txt 文件实现的。

6.结算界面
涉及到 ResultWindow 类，定义如下。

```
class ResultWindow : public QWidget
{
    Q_OBJECT
public:
    explicit ResultWindow(QWidget *parent = nullptr);
    OpenAnimation* openAnimation;
    ShutterAnimation* shutterAnimation;
    QLabel* backContent;
    QPushButton* back;
    QLabel* rating;
    QLabel* cover;
    QLabel* songname;
    QLabel* artist;
    QLabel* score;
    QLabel* scoreTitle;
    QLabel* judgeNum[4]; //0-Perfect 1-Great 2-Good 3-Miss
    QLabel* judgeTitle[4];
    QLabel* maxCombo;
    QLabel* maxComboTitle;
    QLabel* accuracy;
    QLabel* accuracyTitle;
    QLabel* base;
    ~ResultWindow();

public slots:
    void goToSelection();

signals:

};
```

（1）利用 QPalette 和 QPixmap 将相应资源文件作为背景；
（2）在左上角展示相应的曲目信息，包括曲目海报，曲目名称以及曲目作者；
（3）在游戏结束后，会在相应文件夹中读前一个界面的 txt 文件，文件中有 score、perfect、great、good、miss 个数、accuracy 以及 combo 最大数等信息，通过读取 txt 文件中的信息，用 label 在相应位置展示属性以及相应成绩，其中用到了较多的图片资源文件；
（4）在左下角设置了 back 按钮，设置按下时的高亮效果，实现按下 back 按钮到达选歌界面的效果。

三、小组成员分工情况

陈灵钧——SelectionWindow 类，SettingsWindow 类的设计和制作以及部分界面的衔接
虞郑涵——StartWindow 类，ResultWindow 类的设计和制作以及部分界面的衔接
赵大元——GameWindow 类，Note 类，NoteHandler 类，OpenAnimation 类，ShutterAnimation 类的设计和制作和文件的整合

四、项目总结与反思

1.我们小组的项目是音游，一定程度上参照了市面上的音乐游戏，在整体项目的制作过程中我们逐渐学习了对 Qt 软件的使用，对 Qt 软件中不同类、不同界面的关系进行了一定了解，并且学会了简单的信号和槽函数，利用按钮、标签等不同组件构建出了一个音游项目，还是有一定收获的；
2.我们利用 Qt 的特性对类、对象有了更深层次的了解与使用，在 Qt 项目中以不同类的界面相互转换为基础构建了我们的音游项目，同时 Qt 项目将头文件、源文件、界面文件分开存放的方式帮助我们更好的分析一个类的结构，并且可以从不同文件入手分别用间接、直接的方式进行类之间的转化；
3.音游项目的美化是后期的重点，包括图片选取、界面转化中的动画制作等，我们利用网上的资源进行加工排版，形成了有特色的音游；
4.在项目开始时，我们对项目进行了一定的分工，但在各自制作各自部分时并没有很好的统一界面命名以及界面转换方式，导致后期将各个部分拼接起来的时候出现了一定程度的不接合，进而导致最终拼接部分的任务量增加；
5.分工制作的时候对资源的整合效果较差，最终仍然需要将资源整合到同一个资源文件下并

且修改资源路径；
6.在写的过程中没有很好地顾及项目前后的一致性，比如在 NoteHandler 类的设计过程中使用 QTimer::singleShot 导致实现暂停功能十分困难，最终不得不放弃；
7.代码风格不同导致整合代码的时候带来了一些麻烦。