高级语言程序设计

实验报告

南开大学 计算机大类

姓名：张子馨

学号：2413330

班级：2-3

2025 年 5 月 3 日

目录

高级语言程序设计大作业实验报告

一. 作业题目

音乐节奏游戏实现

二. 开发软件

Visual Studio 2017，Qt Creator15.0.1

三. 课题要求

1) 面向对象。

2) 单元测试。

3) 模型部分

4) 验证

四. 主要流程

1．整体流程

实现思路：

1. 主界面

（1）设置静音按钮（左上角按钮）：音乐设置和点击按钮音乐播放状态的改变

```
//成员变量
    QMediaPlayer *bgmPlayer;
    QAudioOutput *audioOutput;
    QPushButton *muteBtn;
 bool isMusicMuted = false;
```

```
//槽函数
void toggleMusic();
```

```
//音乐控制
void MainMenu::setupMusicControls()
{audioOutput = new QAudioOutput(this);
    // 创建播放器
    bgmPlayer = new QMediaPlayer(this);
    bgmPlayer->setAudioOutput(audioOutput);
   ...
    bgmPlayer->play();
    muteBtn = new QPushButton(this);// 创建静音按钮
    muteBtn->setObjectName("muteBtn");
    muteBtn->setFocusPolicy(Qt::NoFocus);
   ...
    // 连接信号槽
    connect(muteBtn, &QPushButton::clicked, this,
&MainMenu::toggleMusic);
    ...
}
```

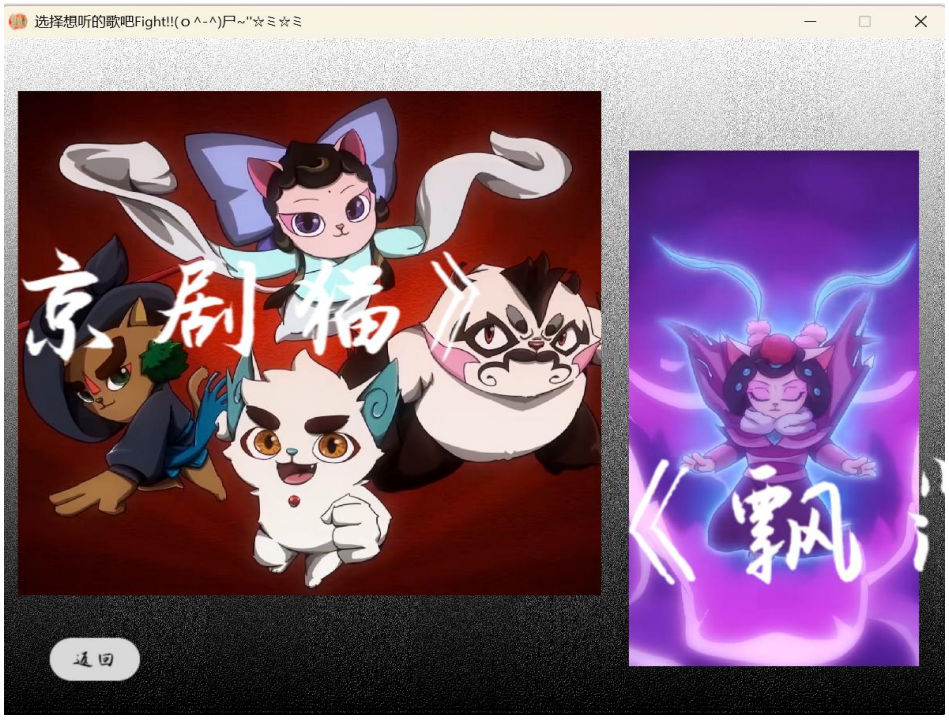
```
void MainMenu::toggleMusic()
{
    //改变静音bool值
    isMusicMuted = !isMusicMuted;

    // 静音控制
    audioOutput->setMuted(isMusicMuted);

    // 更新按钮图标
    muteBtn->setIcon(isMusicMuted
        ? QIcon(":/forzhuye/picture1/jingyin23.png")
        : QIcon(":/forzhuye/picture1/jingyin13.png"));
    ...
}
```

（2）页面切换：点击开始按钮跳转关卡选择界面，这里设置了切换界面的延时

```
//延时跳转选择关卡界面
    connect(startBtn, &MyPushButton::clicked, [=](){
        LevelSelect*levelselect=new LevelSelect;
        audioOutput->setMuted(true);
        this->setFocus();
        this->hide();
        QTimer::singleShot(500,this,[=](){
            levelselect->show();
        });
        //从关卡选择界面返回后操作
        connect(levelselect,&LevelSelect::levelSelectBack,this,[=](){
            levelselect->hide();
            QTimer::singleShot(500,this,[=](){
                this->show();
                //音乐播放状态恢复
                audioOutput->setMuted(isMusicMuted);
            });
        });
    });
```

在页面切换之间不改变 isMusicMuted 的值但要保证跳转页面后主界面音乐必须停止播放，则直接设置播放器静音为 true，在跳转回主界面时再将播放器静音设为 isMusicMuted.

2. 关卡选择页面



（1）背景音设置：除没有静音设置外与主界面一致

（2）返回按钮设置

```
//自定义信号，告诉主场景，点击了返回
signals:
    void levelSelectBack();
```

```
//返回按钮设置
MyPushButton*backBtn=new MyPushButton(":/forselect/picture1/back5.png");
...
//点击返回
connect(backBtn,&MyPushButton::clicked,[=](){
    emit this ->levelSelectBack();
    bgmPlayer2->deleteLater();
});
```

```
//将信号和槽连接
connect(levelselect,&LevelSelect::levelSelectBack,this,[=](){
levelselect->hide();
QTimer::singleShot(500,this,[=](){
        this->show();
         ...
    });
});
```

（3）关卡按钮：共设置两个关卡，点击跳转对应游戏界面

a.设置关卡按钮

```
MyPushButton*leveloneBtn=new MyPushButton(":/forselect/picture1/level14.png");
...
MyPushButton*leveltweBtn=new MyPushButton(":/forselect/picture1/level23.png");
...
```

b.点击关卡跳转游戏界面

在头文件中先初始化游戏场景对象指针，再连接信号与槽

```
//头文件中
//游戏场景的对象指针
 PlayScene*play=NULL;

//.cpp文件中
 connect(leveloneBtn,&MyPushButton::clicked,[=]()
 {
     //进入游戏场景
     ...
     play=new PlayScene(1);//游戏场景的对象指针
     ...
 });

 connect(leveltweBtn,&MyPushButton::clicked,[=]()
 {
     //进入游戏场景
     ...
     play=new PlayScene(2);
     ...
 });
```

3.游戏界面：关卡一和关卡二仅在游戏音乐及猫猫头生成速度上有不同，这里以关卡一示例

（1）返回按钮：与关卡选择按钮设置一致

（2）轨道和按键设置

```
// 轨道方向映射表（轨道号 -> 箭头方向）
    const QMap<int, MyArrow::Direction> TRACK_DIRECTION_MAP = {
        {0, MyArrow::U},
        {1, MyArrow::D},
        {2, MyArrow::L},
        {3, MyArrow::R}
    };

    // 定义按键到轨道的映射表
    const QMap<int, int> keyTrackMap = {
        {Qt::Key_Up,    0}，// 上键对应轨道0
        {Qt::Key_Down，1}，// 下键对应轨道1
        {Qt::Key_Left，2}，// 左键对应轨道2
        {Qt::Key_Right, 3}   // 右键对应轨道3
    };
```

（3）箭头生成

```
//创建箭头
void PlayScene::createArrow(int track)
{
    // 根据轨道号获取对应方向
    MyArrow::Direction dir = TRACK_DIRECTION_MAP.value(track);

    MyArrow *arrow = new MyArrow(dir, track, this);

    ...
}
```

对箭头的后续操作

```
//统一移动所有活动箭头
void PlayScene::moveAllArrows(){...}
//创建箭头
void PlayScene::createArrow(int track){...}
//键盘事件处理
void PlayScene::keyPressEvent(QKeyEvent *event){...}
//碰撞检测，判定是否成功点击按键
void PlayScene::checkCollision(int track){...}
//命中,箭头直接失效消失
void PlayScene::handleHit(MyArrow *arrow){...}
// 未命中，箭头图像改变，停止移动
void PlayScene::handleMiss(MyArrow *arrow){...}
```

2．算法或公式

基于在头文件中已将轨道值、箭头方向、按键一一匹配的映射表，通过玩家的按按键情况判断箭头匹配成功与否。

const int track = keyTrackMap[event->key()];

再将轨道值传入碰撞检测函数

```
//碰撞检测，判定是否成功点击按键
void PlayScene::checkCollision(int track)
{
    const int judgeRange = 1;//判定误差
    bool hit = false;//一开始的撞击状态

    // 只检测对应轨道的箭头
    for(MyArrow *arrow : m_activeArrows) {

        if(arrow->track() != track&&arrow->isActive()) break;

        if(arrow->isActive() &&
            (  m_judgelineY-arrow->y()) > judgeRange)
        {
            handleHit(arrow);
            hit = true;
            break; // 每个按键只命中一个箭头
        }
    }

    if(!hit) handleMiss(m_activeArrows[0]);//保证只与最近箭头匹配
}
```

对命中箭头处理：箭头失效并消失

```
void PlayScene::handleHit(MyArrow *arrow)
{
    arrow->deactivate();

    arrow->hide();

    arrow->deleteLater();
}
```

对未命中箭头处理：箭头图像改变，并在停止运动，2 秒后消失

```
void PlayScene::handleMiss(MyArrow *arrow)
{
    if(arrow->y()<m_judgelineY){
        arrow->setFailed();
        arrow->deactivate();
        m_activeArrows.removeAll(arrow);
        QTimer::singleShot(2000,this,[=](){
            arrow->hide();
            arrow->deleteLater();
        });
    }
}
```

3．单元测试

针对每种游戏操作，测试箭头表现情况是否符合预期。

五.　收获

1.写系统本身自带鼠标和按键事件时要写虚函数重写该事件。

2.学习并练习了信号与槽的连接。

3.学习了计时器的用法。

4.学习了 qt 中动画的写法。