高级语言程序设计

实验报告

南开大学 计算机大类
姓名 沈熙然
学号 2211859
班级 信息安全
2024 年 5 月 13 日

目录

高级语言程序设计大作业实验报告

一. 作业题目

基于 QT 开发的跑酷类游戏《小青蛙爱冒险》

二. 开发软件

QTcreator

三. 课题要求

1.学生自选题目，使用 C++语言完成一个图形化的小程序。
2.图形化平台不限，可以是 MFC、QT 等任何 C++图形化平台。
3.程序内容主题不限，可以是小游戏、小工具等。

四. 主要界面介绍

主窗口（mainwindow）如下：

游戏时候的界面:

背景音乐可以选择播放和暂停。
每次跳起来和打到小怪和顶到上方的问题砖块等都会出现音效。

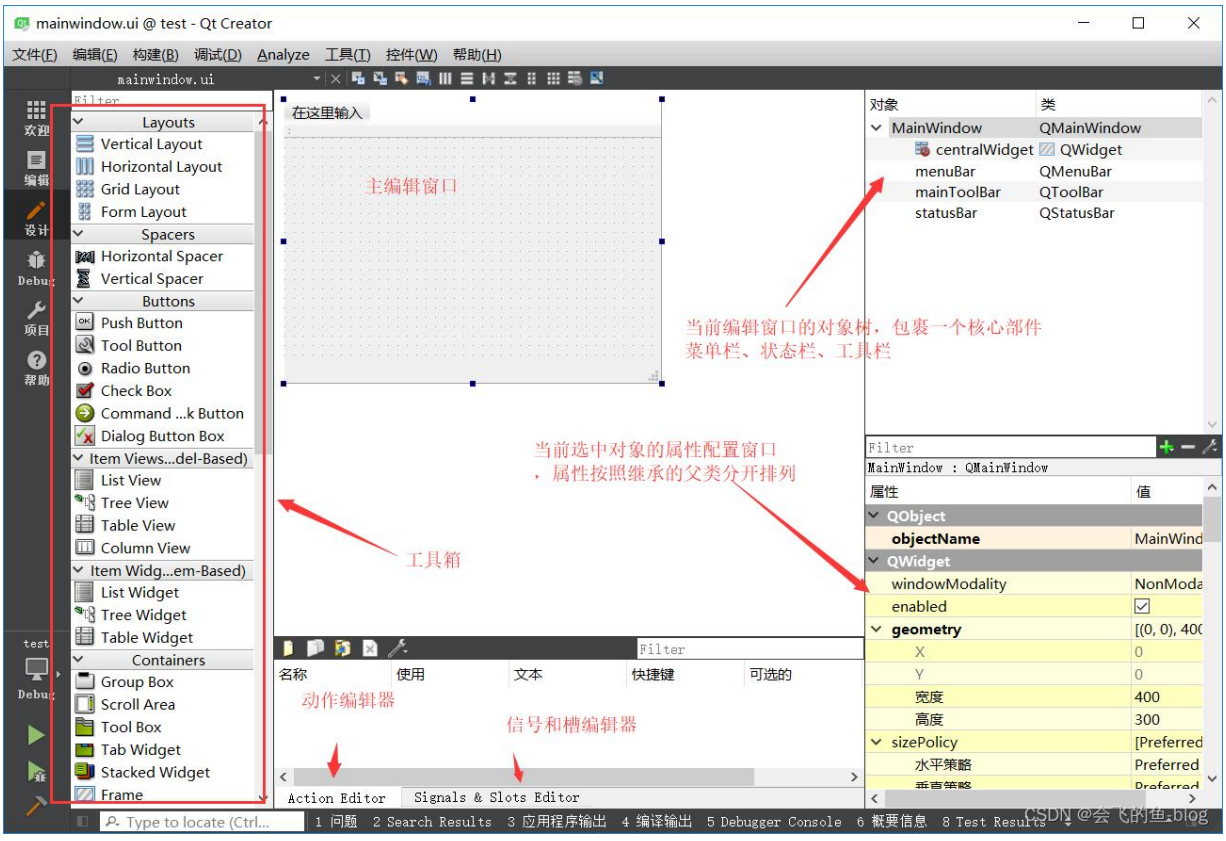
五.　功能实现方法介绍

（1）实现音乐播放功能

```
dj = new DJ(this);//创建播放管理器
 connect(ui->music, &QAction::triggered, this, [=](){
        if (musicEnabled) {
             emit soundeffect("stop");
         } else {
```

```
            emit soundeffect("theme");
        }
        musicEnabled = !musicEnabled; // 切换状态
    });//实现音乐的播放和暂停
```

在 ui 设计中添加了设置音乐开关的选项



进入游戏界面后，同时使用 connect(this, &MainWindow::soundeffect, dj, &DJ::playsoundeffect);emit soundeffect("theme");使得音乐可以播放。

同时在 mainwindow 中定义了 signal 函数“emit sound”，在 dj 中定义了 slot 函数，进而完成对音乐的播放和暂停。

在我写程序的过程中，一直出现音乐的文件解码错误的情况，后来发现是因为必须得转化为 wav 格式，并且必须得设置成低码率。

（2）实现青蛙的跳跃、移动等的代码

设置了一个定时器来处理动画帧的更新。

青蛙的移动：

cat_move() 函数首先检查 cat_can_move 变量是否为真。如果为真，表示青蛙角色可以移

动。
接着根据 cat_stand 变量的值，判断青蛙角色是站立状态还是移动状态：
如果 cat_stand 为真，表示青蛙角色正在站立。在这种情况下，将青蛙角色的水平位置向左移动 22 个单位，并且同步更新 fakecat_x 变量。
如果 cat_stand 为假，表示青蛙角色正在移动。在这种情况下，将青蛙角色的水平位置向右移动 22 个单位，并且同步更新 fakecat_x 变量。
最后，通过调用 cat_rec.moveTo(cat_x, cat_y) 将青蛙角色的位置更新到新的坐标 (cat_x, cat_y) 上。

青蛙的跳跃：

如果青蛙的当前垂直位置大于等于跳跃目标高度 cat_jump_height 且处于上升状态 (cat_up == true)，则向上移动青蛙角色并更新其垂直位置。如果达到了跳跃目标高度，则停止向上移动。
如果青蛙的当前垂直位置小于最小高度 cat_height_min 且处于下降状态（cat_up == false）并且没有发生下落阶段的情况（cat_down == 0），则向下移动青蛙角色并更新其垂直位置。如果达到了最小高度，则标记跳跃结束，并且重置跳跃相关的状态。
执行类似于青蛙角色的跳跃逻辑，但在计算最小高度时，加上了额外的偏移量 BIAS。
如果青蛙当前正在下落（cat_down != 0），则将其标记为可以自由跳跃，并且更新跳跃目标高度为当前垂直位置减去 270，这样确保青蛙在下落阶段仍然可以执行跳跃。
由此实现了青蛙角色在游戏中的跳跃行为，包括了在不同情况下的上升、下降和跳跃结束的处理。

（3）各种物体的设置（以洞穴设置为例）

```
    }
}
void MainWindow::set_hole()//设置洞穴
    {
        hole[0].hole_x=2520;
        hole[0].hole_y=HEIGHT-GreenGround_Height-GROUND_WIDTH-15;
        hole[1].hole_x=2520+Hole_LENGTH-GreenGround_Width;
        hole[1].hole_y=HEIGHT-GreenGround_Height-GROUND_WIDTH-15;
        hole[2].hole_x=4550;
        hole[2].hole_y=HEIGHT-GreenGround_Height-GROUND_WIDTH-15;
        hole[3].hole_x=4550+Hole_LENGTH-GreenGround_Width;
        hole[3].hole_y=HEIGHT-GreenGround_Height-GROUND_WIDTH-15;

        hole[4].hole_x=2520;
        hole[4].hole_y=HEIGHT-GreenGround_Height-GROUND_WIDTH+26-GreenGround_Height;
        hole[5].hole_x=2520+Hole_LENGTH-GreenGround_Width;
        hole[5].hole_y=HEIGHT-GreenGround_Height-GROUND_WIDTH+26-GreenGround_Height;
        hole[6].hole_x=4550;
        hole[6].hole_y=HEIGHT-GreenGround_Height-GROUND_WIDTH+26-GreenGround_Height;
        hole[7].hole_x=4550+Hole_LENGTH-GreenGround_Width;
        hole[7].hole_y=HEIGHT-GreenGround_Height-GROUND_WIDTH+26-GreenGround_Height;

        hole[8].hole_x=2520;
        hole[8].hole_y=HEIGHT-GreenGround_Height-GROUND_WIDTH+BIAS-1;
        hole[9].hole_x=2520+Hole_LENGTH-GreenGround_Width;
        hole[9].hole_y=HEIGHT-GreenGround_Height-GROUND_WIDTH+BIAS-1;
        hole[10].hole_x=4550;
        hole[10].hole_y=HEIGHT-GreenGround_Height-GROUND_WIDTH+BIAS-1;
        hole[11].hole_x=4550+Hole_LENGTH-GreenGround_Width;
        hole[11].hole_y=HEIGHT-GreenGround_Height-GROUND_WIDTH+BIAS-1;

        hole[12].hole_x=2520;
        hole[12].hole_y=HEIGHT-GreenGround_Height-GROUND_WIDTH+BIAS+30-GreenGround_Height;
        hole[13].hole_x=2520+Hole_LENGTH-GreenGround_Width;
        hole[13].hole_y=HEIGHT-GreenGround_Height-GROUND_WIDTH+BIAS+30-GreenGround_Height;
        hole[14].hole_x=4550;
        hole[14].hole_y=HEIGHT-GreenGround_Height-GROUND_WIDTH+BIAS+30-GreenGround_Height;
```

是通过设置数组，然后设置坐标的位置，一个一个显示出来。

每一个界面显示的组件都有自己的类。

```
Hole::Hole(QWidget *parent) : QWidget(parent)
{
    hole_pix.load(GreenGround_PICTURE);
    hole_rec.setWidth(24);
    hole_rec.setHeight(96);
    fire_hole_rec.setWidth(80);
    fire_hole_rec.setHeight(80);
}

void Hole::fire_moving()
{
    timer6.start(fire_TIME);
    connect(&timer6,&QTimer::timeout,[=]()                    Pass a context object
    {
        fire_hole_pix.load(Rocket_crash_PICTURE);
        fire_hole_pix = fire_hole_pix.scaled(100,100);
        if(fire_up==true)
        {
            fire_hole_y-=10;//单次上升10
            if(fire_hole_y<=fire_up_max)
            {
                fire_up=false;
                fire_down=true;
            }
        }
        if(fire_down==true)
        {
            fire_hole_y+=10;//单次下降10
            if(fire_hole_y>=fire_down_max)
            {
                fire_up=true;
                fire_down=false;
            }
        }
        fire_hole_rec.moveTo(fire_hole_x,fire_hole_y);
    });
}
```

```
class Hole : public QWidget
{
    Q_OBJECT
public:
    explicit Hole(QWidget *parent = nullptr);
    QPixmap hole_pix;
    QPixmap fire_hole_pix;
    int hole_x;
    int hole_y;
    int fire_hole_x;
    int fire_hole_y;
    int fire_up_max;
    int fire_down_max;
    bool fire_up=true;
    bool fire_down=false;

    QRect hole_rec;
    QRect fire_hole_rec;

    QTimer timer6;
    void fire_moving();

signals:

public slots:
};

#endif // HOLE_H
```

Hole 类的代码如上图。

（4）实现游戏结束界面的动态播放

```cpp
#include "winwindow.h"
#include "config.h"
#include <QDebug>
#include <QCoreApplication>
Winwindow::Winwindow(QWidget *parent) : QMainWindow(parent)
{
    setFixedSize(947, 720);
    win_pix.load(WIN_PIC);//游戏结束图片

    timer.start(15);

    connect(&timer,&QTimer::timeout,this,[=](){
        update();
    });
}

void Winwindow::paintEvent(QPaintEvent *event){
    QPainter painter(this);

    painter.drawPixmap(0, y--, win_pix);
    if(y == -1330)
      QCoreApplication::quit();
}

```

六. 收获

(1) 提高了我的自学能力和信息检索能力。
(2) 提高了我的 debug 能力。
(3) 对于 c++的图形化编程有了更加深刻的理解，也对 qt 的用法有了一定的认识。
(4) 过程中用到的类的介绍。（内容来自网络查阅）

QApplication：该类管理 GUI 程序的控制流和主要设置，是基于 QWidget 的，为此特化了 QGuiApplication 的一些功能，处理 QWidget 特有的初始化和结束收尾工作。对于使用了 Qt 的任何 GUI 程序来说，不管何时何地有多少个 Window，但只有一个 QApplication 对象，如果不是基于 QWidget 的程序，相应的则使用 QGuiApplication，后者不依赖于 Widget 特有的库。在 main 函数中通过它获得了用户界面的高度宽度，进而使得界面设置居中。

QResource：程序中用 QResource 类加载 QResource::registerResource("/path/resource.rcc")，进而动态加载资源。

QObject：只有继承了 QObject 类的类，才具有信号槽的能力。所以，为了使用信号槽，必须继承 QObject。凡是 QObject 类（不管是直接子类还是间接子类），都应该在第一行代码写上 Q_OBJECT。不管是不是使用信号槽，都应该添加这个宏。这个宏的展开将为我们的类提供信号槽机制、国际化机制以及 Qt 提供的不基于 C++ RTTI 的反射能力。因此，如果你觉得你的类不需要使用信号槽，就不添加这个宏，就是错误的。其它很多操作都会依赖于这个宏。

QMainWindow：是一个为用户提供主窗口程序的类，包含一个菜单栏（menu bar）、多个工具栏(tool bars)、多个停靠部件(dock widgets)、一个状态栏(status bar)及一个中心部件(central widget)，是许多应用程序的基础，如文本编辑器，图片编辑器等。

QWidget：每个 Widget 的构造函数接收一或两个参数。如果 QWidget *parent =nullptr：新 Widget 的 Parent。如果它是 nullptr（这也是默认选项），那么新 Widget 将会是一个 Window；

如果不是，那么它将成为 parent 的子孙，并且将限制在 parent 的几何空间内（除非指明 Qt::Window 为 WindowFlag）。

namespace Ui：可能让人有点摸不着头脑，这是因为 qt 把 ui 相关的类单独独立了出来，但类名相同，禁用 namespace 区别，声明 namespace Ui 是因为要调用 Ui 中的 MainWindow，此 MainWindow 非彼 MainWindow，后面涉及的*ui 指针会调用它！构造时在堆上 new 了个 Ui 域中的 MainWindow，并调用 setupUI

QGraphicsScene：是 Qt 中用于管理 2D 图形项（QGraphicsItem）的场景类。它充当了图形项的容器，负责管理图形项的布局、渲染、事件处理等。QGraphicsScene 可以看作是一个虚拟的画布，上面可以放置多个图形项，并且可以对这些图形项进行管理和操作。

QPixmap ：是 Qt 中用于处理图像的类，它是基于屏幕的图像表示方式，可以用于在 Qt 应用程序中显示图像、图标和背景。你可以使用 QPixmap 的构造函数或者 load() 函数来加载图像。构造函数可以直接传递图像文件的路径，load() 函数则需要在加载前设置文件路径。

QTimer：只需创建一个 QTimer 类对象，然后调用其 start() 函数开启定时器，此后 QTimer 对象就会周期性的发出 timeout() 信号。将其 timeout()信号连接到适当的插槽，并调用 start()。从那时起，它将以恒定的间隔发出 timeout()信号。