程序设计综合实践

报告

|  |  |
| --- | --- |
| 班级 | 物联网工程2016级2班 |
| 姓名 | 郝宗寅 |
| 学号 | 201601061408 |
| 成绩 |  |

2017年7月17日

山东科技大学

# 1. 相关技术与工具介绍

## 1.1 信号与槽

就我个人来理解，信号槽机制与Windows下消息机制类似，消息机制是基于回调函数，Qt中用信号与槽来代替函数指针，使程序更安全简洁。

信号和槽机制是 Qt 的核心机制，可以让编程人员将互不相关的对象绑定在一起，实现对象之间的通信。

* **信号**

当对象改变其状态时，信号就由该对象发射 (emit) 出去，而且对象只负责发送信号，它不知道另一端是谁在接收这个信号。这样就做到了真正的信息封装，能确保对象被当作一个真正的软件组件来使用。

* **槽**

用于接收信号，而且槽只是普通的对象成员函数。一个槽并不知道是否有任何信号与自己相连接。而且对象并不了解具体的通信机制。

* **信号与槽的连接**

所有从 QObject 或其子类 ( 例如 Qwidget ) 派生的类都能够包含信号和槽。因为信号与槽的连接是通过 QObject 的 connect() 成员函数来实现的。

connect(sender, SIGNAL(signal), receiver, SLOT(slot));

其中 sender 与 receiver 是指向对象的指针，SIGNAL() 与 SLOT() 是转换信号与槽的宏。

## 1.2 Qt事件机制

Qt程序是事件驱动的, 程序的每个动作都是由幕后某个事件所触发.。

Qt事件的发生和处理成为程序运行的主线，存在于程序整个生命周期。

本次实验用的事件机制主要为键盘、鼠标事件机制，重写了图形视图框架提供的控制函数*keyPressEvent、keyReleaseEvent、mouseDoubleClickEvent等函数实现方向的控制和菜单的调用。*

## 1.3 QTimer控制时间

 Qt中有两种方法来使用定时器，一种是定时器事件，另一种是使用信号和槽。本次实验主要使用了信号和槽的关联来实现时间的控制，例如connect(timer,&QTimer::timeout, this, &Widget::timerUpdate)；

## 1.4 随机数

关于随机数，在Qt中是使用qrand()和qsrand()两个函数实现的。在使用qrand()函数产生随机数之前，一般要使用qsrand()函数为其设置初值，如果不设置初值，那么每次运行程序，qrand()都会产生相同的一组随机数。为了每次运行程序时，都可以产生不同的随机数，我们要使用qsrand()设置一个不同的初值。这里使用了QTime类的secsTo()函数，它表示两个时间点之间所包含的秒数，比如qsrand(QTime(0, 0, 0).secsTo(QTime::currentTime()));然后直接使用qrand（）产生随机数就可以了

## 1.5 使用QPainter绘图

这次实验主要用到QPainter绘制图片的功能。   Qt提供了四个类来处理图像数据：QImage、QPixmap、QBitmap和QPicture，它们都是常用的绘图设备。其中QImage主要用来进行I/O处理，它对I/O处理操作进行了优化，而且可以用来直接访问和操作像素；QPixmap主要用来在屏幕上显示图像，它对在屏幕上显示图像进行了优化；QBitmap是QPixmap的子类，用来处理颜色深度为1的图像，即只能显示黑白两种颜色；QPicture用来记录并重演QPainter命令。这次实验主要用到了QPainterPath，添加图片可以在构造函数中完成，或者调用load函数，调整图片大小可以用scale函数，绘制图片可以调用QPainter的drawPixmap来实现

## 1.6 图形视图框架

本次实验最关键的技术。

图形视图可以对大量定制的2D图形项进行管理和相互作用。视图部件可以让所有图形项可视化，它还提供了缩放和旋转功能。

整个图形视图结构主要包含三部分：场景（Scene）、视图（View）和图形项（Item），它们分别对应 QGraphicsScene 、QGraphicsView 、QGraphicsItem三个类。一切绘图都是在场景中进行的，通过view显示，item则是不同的图形项。

本次实验用了自定义图形项，重写了其boundingrect函数、shape函数以及paint函数，实现不同的显示效果。同时使用了图形视图框架提供的键盘鼠标事件函数和、画函数以及碰撞检测函数，为所有类添加了name函数标识结合系统的碰撞函数来进行碰撞检测， 并重写了advance函数实现了各种动画效果，使用了Qt封装好的各函数优化图片的缓冲机制，实现图片的反走样优化了图片的显示

## 1.7 主要工具

Qt creator、Qt帮助文档、百度

# 2. 项目设计

## 2.1 功能设计

## C:\Documents and Settings\Administrator\Desktop\Image2.EMF2.2 静态结构设计

C:\Documents and Settings\Administrator\Desktop\Image2.EMF

## 2.3 类的详细设计

核心类中关键方法的算法流程（可用活动图、顺序图或程序流程图）。

## 2.4 界面设计

菜单界面采用最经典的qt ui设计师界面，游戏界面采用无边框设计更美观

# 项目实现

控制中心global：

#ifndef SPACE\_H

#define SPACE\_H

#include "planefactory.h"

#include "playerPlane.h"

#include "bloodsupply.h"

#include "bombsupply.h"

#include "bulletsupply.h"

#include "randomizer.h"

#include "menuwidget.h"

class Space : public QGraphicsView

{

Q\_OBJECT

public:

Space(QWidget \*parent = 0);

void init();

void bloodsupply();//生成血包

void bombsupply();//生成炸弹包

void bulletsupply();//生成子弹包

void enemys();//生成敌人

protected:

void *mouseDoubleClickEvent*(QMouseEvent \*event);//鼠标双击事件

private:

QGraphicsScene \*m\_scene;//场景

QTimer \*m\_timer;//时间控制

PlayerPlane \*m\_player;//玩家

uint m\_scores, m\_level, m\_step;//得分决定level，level决定游戏难度，step控制物体出现频率

bool boss;//boss战标识

bool isRunning;//程序是否运行

signals:

void sig\_menu();//调用菜单的信号

protected slots:

void slt\_newGame();//新游戏

void slt\_playerDead();//玩家死亡

void slt\_startGame();//开始游戏

void slt\_pauseGame();//暂停游戏

void slt\_updata();//刷屏函数

void slt\_addscore(int);//得分增加的槽函数，与敌机中的score信号关联

void slt\_menu();//产生菜单的槽函数

};

#endif // SPACE\_H

#include "space.h"

Space::Space(QWidget \*parent) : QGraphicsView(parent), boss(false), isRunning(false) {

m\_scene = new QGraphicsScene;//创建新场景，并设置大小，然后把场景加入视图

m\_scene->setSceneRect(0, 0, SCENEWIDTH, SCENEHEIGHT);

setScene(m\_scene);

setWindowFlags(Qt::FramelessWindowHint);//无菜单栏

setVerticalScrollBarPolicy(Qt::ScrollBarAlwaysOff);//关闭垂直滚动条

setHorizontalScrollBarPolicy(Qt::ScrollBarAlwaysOff);//关闭水平滚动条

setCacheMode(QGraphicsView::CacheBackground);//设置缓冲模式

setViewportUpdateMode(QGraphicsView::BoundingRectViewportUpdate);

setOptimizationFlags(QGraphicsView::DontClipPainter | QGraphicsView::DontSavePainterState);

setRenderHint(QPainter::Antialiasing);//反走样，提高画图质量

QPixmap pix(SCENEWIDTH, SCENEHEIGHT);//按照场景大小添加背景

pix.load(background);

QPixmap temp = pix.scaled(SCENEWIDTH, SCENEHEIGHT, Qt::KeepAspectRatioByExpanding);

setBackgroundBrush(temp);

m\_timer = new QTimer;

connect(m\_timer, SIGNAL(timeout()), this, SLOT(slt\_updata()));//关联时间和刷屏

connect(this, SIGNAL(sig\_menu()), this, SLOT(slt\_menu()));

init();

}

void Space::init() {

m\_scores = 0, m\_bloods = PLAYERPLANEBLOOD, m\_bombs = 0, m\_level = 1, m\_step = 0;

QPixmaps t;//创建玩家是使用图片组，可以实现动态效果

t.append(QPixmap(playerplane0));

t.append(QPixmap(playerplane1));

t.append(QPixmap(playerplane2));

t.append(QPixmap(playerplane3));

m\_player = new PlayerPlane(PLAYERPLANEBLOOD, PLAYERPLANESIZE, PLAYERPLANESIZE, PLAYERPLANESPEED, t, m\_scene);

m\_player->setFocus();//设置焦点，从而可以用键盘控制

connect(m\_player, SIGNAL(sig\_fall()), this, SLOT(slt\_playerDead()));

}

void Space::slt\_newGame()

{

m\_scene->clear();//清屏

init();

m\_timer->start(50);//计时开始，50毫秒刷一次屏

isRunning = true;

}

void Space::slt\_playerDead()

{

m\_timer->stop();//计时停止

isRunning = false;

}

void Space::slt\_startGame()

{

if (isRunning) m\_timer->start(50);

}

void Space::slt\_pauseGame()

{

if (isRunning) m\_timer->stop();

}

void Space::slt\_updata()

{

m\_scene->advance();//系统封装好的动画函数，第一步告诉所有item将要移动，第二步让所有item移动，优化的很好

m\_step++;

if (m\_step % 50 == 0 && !boss) {

enemys();

}

if (m\_step % 233 == 0) {

bulletsupply();

}

if (m\_step % 666 == 0) {

bombsupply();

m\_step = 0;

}

if (m\_step % 2333 == 0) {

bloodsupply();

}

}

void Space::slt\_addscore(int score)

{

m\_scores += score;

if (score == BOSSSCORE) {//游戏目前只有一关，打完boss后一切归0

boss = false;

m\_level = 1;

m\_scores = 0;

}

if (m\_scores != 0 && m\_scores % 50 == 0) {

m\_level++;

if (m\_level == 7) m\_level = 1;

}

}

void Space::bloodsupply()

{

QPixmaps t;

t.append(QPixmap(bloodsupplypix));

BloodSupply \*bloodsupply = new BloodSupply(BLOODSUPPLYSIZE, BLOODSUPPLYSIZE, BLOODSUPPLYSPEED, t, m\_scene);

qreal x = Randomizer::creat(SCENEWIDTH - BLOODSUPPLYSIZE);

bloodsupply->setPos(x, 0);

}

void Space::bombsupply()

{

QPixmaps t;

t.append(QPixmap(bombsupplypix));

BloodSupply \*bombsupply = new BloodSupply(BOMBSUPPLYSIZE, BOMBSUPPLYSIZE, BOMBSUPPLYSPEED, t, m\_scene);

qreal x = Randomizer::creat(SCENEWIDTH - BOMBSUPPLYSIZE);

bombsupply->setPos(x, 0);

}

void Space::bulletsupply()

{

QPixmaps t;

t.append(QPixmap(bulletsupplypix));

uint x = Randomizer::creat(SCENEWIDTH - BULLETSUPPLYSIZE);

BulletSupply \*bulletsupply = new BulletSupply(BULLETSUPPLYSIZE, BULLETSUPPLYSIZE, BULLETSUPPLYSPEED, t, m\_scene);

bulletsupply->setPos(x,0);

}

void Space::enemys()

{

if (m\_level == 6) {

PlaneFactory::BossPlanes bossplanes = PlaneFactory::bcreator(1, scene());

foreach (Boss \*b, bossplanes) {

int x = Randomizer::creat(SCENEWIDTH - BOSSSIZE);

b->setPos(x, 0);

connect(b, SIGNAL(sig\_score(int)), this, SLOT(slt\_addscore(int)));

}

boss = true;

}

else {

PlaneFactory::FishPlanes fishplanes = PlaneFactory::fcreator(m\_level, scene());

foreach (Fish \*f, fishplanes) {

int z = Randomizer::creat(6);

if (z == 1) {

int y = Randomizer::creat(SCENEHEIGHT / 2 - FISHSIZE);

f->setPos(0, y);

}

else if (z == 2) {

int y = Randomizer::creat(SCENEHEIGHT / 2 - FISHSIZE);

f->setPos(SCENEWIDTH - FISHSIZE, y);

}

else {

int x = Randomizer::creat(SCENEWIDTH - FISHSIZE);

f->setPos(x, 0);

}

connect(f, SIGNAL(sig\_score(int)), this, SLOT(slt\_addscore(int)));

}

}

}

void Space::slt\_menu()

{

if (isRunning) {

slt\_pauseGame();

QScopedPointer<MenuWidget> w(new MenuWidget(true, this));//新建菜单

connect(w.data(), SIGNAL(sig\_newGame()), this, SLOT(slt\_newGame()));

connect(w.data(), SIGNAL(sig\_quit()), this, SLOT(close()));

w->setModal(true);

w->show();

w->*exec*();

slt\_startGame();

}

else {

QScopedPointer<MenuWidget> w(new MenuWidget(false, this));

connect(w.data(), SIGNAL(sig\_newGame()), this, SLOT(slt\_newGame()));

connect(w.data(), SIGNAL(sig\_quit()), this, SLOT(close()));

w->setModal(true);

w->show();

w->*exec*();

}

}

void Space::*mouseDoubleClickEvent*(QMouseEvent \*event) {

event->accept();

emit sig\_menu();//双击屏幕发射菜单调用信号

}

所有飞行物体的基类：

#ifndef FLYER\_H

#define FLYER\_H

#include <QtWidgets>

#include "global.h"

typedef enum {UP, DOWN, LEFT, RIGHT} CHECK\_FLAG;

typedef QList<QPixmap> QPixmaps;//图片组

class Flyer : public QGraphicsObject

{

Q\_OBJECT//为了使用系统提供的一些函数而添加的宏

public:

Flyer(qreal w, qreal h, qreal speed, const QPixmaps &pixs, QGraphicsScene \*scene, QGraphicsItem \*parent = 0)

: QGraphicsObject(parent), m\_w(w), m\_h(h), m\_speed(speed), m\_pixpos(0), m\_step(0) {

for (int i = 0; i < pixs.size(); i++) {//按照规定大小缩放图片，并把一组图片添加到m\_pixs

QPixmap temp(pixs.at(i)), t;

t = temp.scaled(m\_w, m\_h, Qt::KeepAspectRatioByExpanding);

m\_pixs.append(t);

}

scene->addItem(this);//在创建物体时直接添加到场景中

}

virtual ~*Flyer*() {}

virtual int *name*() const = 0;//进行碰撞判断时需要获得每个类的名字

virtual void *posLost*() = 0;//出界销毁，其实可以用fall代替

virtual void *doCollide*() = 0;//碰撞

virtual void *fall*() = 0;//坠落

bool checkPos(CHECK\_FLAG flag)//出界检测

{

bool ok = false;

switch (flag) {

case UP:

if (scenePos().ry() >= -m\_h) ok = true;

break;

case DOWN:

if (scenePos().ry() <= SCENEHEIGHT) ok = true;

break;

case LEFT:

if (scenePos().rx() >= -m\_w) ok = true;

break;

case RIGHT:

if (scenePos().rx() <= SCENEWIDTH) ok = true;

break;

}

return ok;

}

protected:

qreal m\_w, m\_h, m\_speed, m\_pixpos;//物体大小 & 速度 & 图片位置 & 动画控制

uint m\_step;

QPixmaps m\_pixs;

};

#endif // FLYER\_H

程序特色代码（截取的一些片段）：

#define ROTATE 57.26

m\_angle(angle),

xSpeed(::cos(m\_angle / ROTATE) \* m\_speed),

ySpeed(::sin(m\_angle / ROTATE) \* m\_speed)

//通过速度和角度计算x轴速度和y轴速度，可以使物体朝任意方向移动，实现炫酷效果

# 4. 实训总结与体会

## 4.1 遇到的主要问题与解决方法

正文内容……

## 4.2 学习心得

开始时因为Qt教程杂乱，各种标准不一很少学到到东西，网上又没有模板，根本不知道如何实现这个程序。但在百度翻了不下100页之后终于有了进展，Qt社区里Qt快速入门的基础知识让我了解了要用图形视图框架来实现我的游戏，Qt帮助文档和谷歌的帮助让我学会使用各种封装好的函数实现各种功能，在程序可以运行的基础上，我又对其做了不下10次从头到尾的改变，让他有了抽象类，有了工厂，实现了各种效果，而抽象类和工厂是我平时做c++时想都不敢想的（一用就出错），一口气改千行代码也是我从未经历过的，可以说这次实验极大提高了我的面向对象编程的技能。但我觉得我最大的收获还是通过自己的努力搞懂了一个从来没有用过的程序，在杂乱的资料中整理出自己需要的知识，在没有前例的情况下总结出了自己的模板，并把它写成了博客，自己的学习能力得到了提高。

## 4.3 建议与意见

最后有点赶工，有些细节没处理好，而且程序只是一个模板，各种物体类型比较单一。主要问题还是效率低下，希望以后能提高学习效率。



图1 集装箱委托业务领域模型

表1 订舱信息表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名 | 编码 | 数据类型 | 长度 | 键 |
| 订舱id | Booking\_id | Varchar | 20 | Pk |
| 委托编号 | Booking\_entrust\_id | Varchar | 20 | Fk |
| 航线信息 | Booking\_route\_id | Varchar | 20 | Fk |
| 船id | Booking\_ship\_id | Varchar | 20 | Fk |
| 创建日期 | Booking\_enterdate | Date |  |  |