南开大学 计算机大类

姓名 马昊洋

学号 2413757

班级 网安三班

2025年5月14日

高级语言程序设计

实验报告

目录

[高级语言程序设计大作业实验报告 1](#_Toc21286)

[一. 作业题目 1](#_Toc29969)

[二. 开发软件 1](#_Toc18364)

[三. 课题要求 1](#_Toc742)

[四. 主要流程 1](#_Toc25240)

[1． 设计思路 2](#_Toc20452)

[2． 整体流程 2](#_Toc25354)

[3． 算法重点 2](#_Toc25354)

[五. 代码迭代 3](#_Toc28265)

[六. 收获 3](#_Toc23116)

[1．待优化点 3](#_Toc10799)

[2．代码亮点 3](#_Toc16275)

1. **作业题目**

基于飞机大战的底层逻辑开发的小游戏

1. **开发软件**

Visual Studio 2022，QT Creator

1. **课题要求**
2. 使用C++语言；
3. 开发图形化小程序，内容不限。
4. **主要流程**
5. 设计思路：

(1) 面向对象设计

游戏对象基类：所有游戏元素（玩家角色、掉落物、匕首等）继承自公共基类（QObject和QGraphicsPixmapItem），统一管理位置、速度、碰撞检测等属性。

模块化组件：

玩家控制：处理键盘输入（WASD移动，空格射击）。

敌机生成：随机类型、位置和移动路径。

碰撞系统：检测匕首、掉落物、玩家角色间的碰撞。

分数与生命值：实时更新并显示在UI上。

(2) Qt框架利用

QGraphicsScene：用于2D游戏场景管理，实现对象的渲染、层级和碰撞检测。

信号与槽机制：处理事件（如按键触发、敌机销毁信号）。

QTimer定时器：控制游戏循环（帧刷新、敌机生成频率）。

(3) 资源管理

图像与音效：通过QPixmap加载素材（飞机、子弹、背景），QFont管理文本效果

内存优化：对象池技术复用子弹和敌机对象，避免频繁创建/销毁。

1. 整体流程

(1) 初始化阶段：

创建场景（QGraphicsScene）。

加载资源（图片、音效），初始化玩家飞机、背景。

启动主游戏定时器（如60FPS）。

(2) 游戏主循环：

输入处理：监听键盘事件，更新玩家位置或触发射击。

敌机生成：定时器随机生成敌机（类型、位置、速度）。

对象更新：

移动子弹和敌机。

检测碰撞（玩家vs敌机、子弹vs敌机）。

渲染：Qt自动处理场景中对象的绘制。

(3)游戏结束判断：

玩家生命值归零时，显示结束界面，重置游戏状态。

1. 算法重点

(1) 碰撞检测优化

矩形检测：使用QGraphicsItem进行粗略检测；

性能问题：大量对象时需优化。

(2) 敌机AI与路径生成

随机运动：各掉落物生成的频率、位置应随机；

差异化行为：Boss与普通掉落物的移动逻辑不同。

(3) 对象池管理

复用对象：避免频繁创建/销毁子弹和敌机，需维护“活跃”和“空闲”列表，处理对象状态重置。

(4) 帧率同步

定时器精度：QTimer的精度依赖系统，可能需手动计算帧间隔（delta time）确保不同硬件下速度一致。

1. **代码迭代**

1.0版本：仅能实现基本的类飞机大战的功能：操控角色（dog）移动、发射子弹（匕首bullet）、有敌机（蘑菇dropment、蜜饼pie）生成、碰撞时消失、分数（饱腹值score、健康值health）变化；

2.0版本：添加了发射匕首的条件：消耗饱腹值；添加了在饱腹值>50时角色发生变化的机制；

3.0版本：添加了在饱腹值>100时出现boss的功能，并设计了相对应的移动模式和攻击行为；添加了boss的特殊子弹cat；添加了新的胜利条件.

1. **收获**

1、待优化点：

(1) 碰撞检测优化

问题：传统遍历所有对象的碰撞检测（O(n²)复杂度）在对象数量多时性能下降。

解决方案：

使用QGraphicsScene内置的collidingItems()或collidesWithItem()方法，利用Qt的边界矩形（Bounding Rect）或像素级碰撞检测。

优化思路：视频中可能通过分层检测（如先粗测矩形重叠，再细测像素）减少计算量。

(2)对象池管理（内存优化）

问题：频繁创建/销毁子弹、敌机对象会导致内存碎片和性能开销。

解决方案：

对象池模式：预生成对象队列（如QList<Bullet>），使用时激活，销毁时标记为“空闲”而非真正删除，即伪代码：对象池复用子弹

Eg. void Player::shoot() {

Bullet bullet = m\_bulletPool.getInactiveBullet(); // 从池中获取空闲子弹

if (bullet) {

bullet>setPos(this>x(), this>y());

bullet>setActive(true); // 激活子弹

}

}

(3)敌机AI与路径生成

难点：实现多样化的敌机移动路径（直线、曲线、追踪玩家等）。

实现方法：

贝塞尔曲线：Boss敌机可能使用二次/三次贝塞尔曲线生成平滑路径。

状态机：敌机行为切换（如巡逻→攻击→逃跑）通过有限状态机（FSM）管理。

(4)帧率同步与游戏循环

问题：QTimer的定时精度依赖系统，可能导致不同硬件下游戏速度不一致。

解决方案：

计算delta time（帧间隔时间），动态调整对象位移：

Eg. void GameScene::update() {

qint64 currentTime = QDateTime::currentMSecsSinceEpoch();

qreal deltaTime = (currentTime m\_lastFrameTime) / 1000.0;

m\_lastFrameTime = currentTime;

// 根据deltaTime移动对象

m\_player>move(m\_player>speed() deltaTime);

}

(5)高效应用Qt框架：可利用信号与槽解耦游戏事件（如bulletHitEnemy信号触发分数更新和爆炸动画）。

(6)内存安全：通过QPointer或QScopedPointer管理动态对象，防止内存泄漏。

(7)扩展性设计:

关卡系统：通过QStateMachine实现不同游戏状态（开始菜单、战斗中、游戏结束）。

Mod支持：预留接口允许外部加载敌机或武器配置（如读取XML/JSON定义新敌机类型）。

2、代码亮点

(1)资源管理方面

纹理与音效：使用QPixmapCache缓存常用图片，避免重复加载；音效通过QSoundEffect异步播放。

(2)代码结构设计

组件化：将游戏逻辑拆分为独立类（如Player、Enemy、Bullet），符合单一职责原则。

配置文件：使用头文件GameSetting管理游戏参数（如敌机速度、生成频率），便于调整平衡性。