高级语言程序设计

实验报告

南开大学 工科试验班

姓名 宗子路

学号 2411811

班级 0978

2025 年 5 月 11 日

目录

高级语言程序设计大作业实验报告

基于 EasyX 图形库的 2D 生存类游戏开发

一、作业题目

桌面端生存射击类游戏开发

具体描述：本项目为一款 2D 生存射击游戏，玩家通过键盘控制角色移动，自动发射子弹攻击敌人，在持续生成的敌人围攻中生存并积累分数。游戏包含主菜单界面、实时战斗系统、得分统计等功能模块。

二、开发软件与环境

编程语言　　: C++11

图形库　　: EasyX

开发工具　　: Visual Studio 2022

多媒体支持　: Windows Multimedia API

操作系统　　: Windows 11

三、课题要求

设计并实现一个简单的“幸存者”类游戏，玩家控制一个角色在地图上移动，角色自动攻击靠近的敌人。游戏的目标是在不断生成的敌人围攻下生存尽可能长的时间。

核心要求：

1.玩家控制：实现角色移动与自动攻击

2.敌人系统：构建敌人 AI 生成与追击系统

3.子弹设计：.开发子弹环绕武器机制

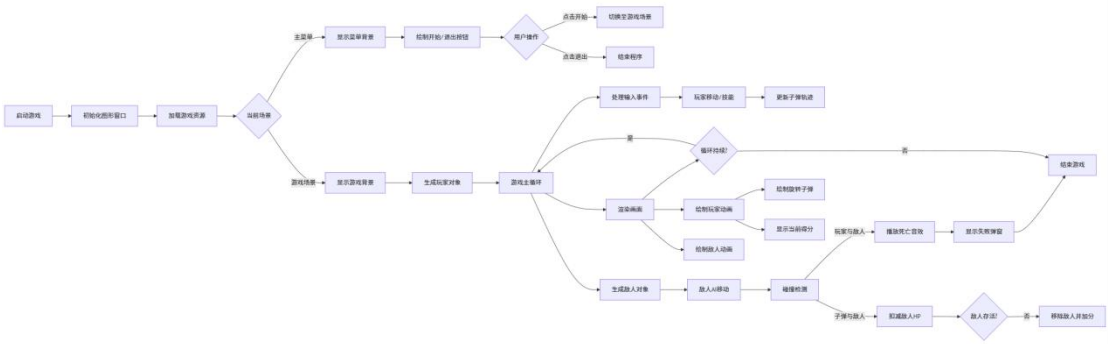
4.图形界面交互:设计主菜单界面与按钮交互

5.得分检测：实现碰撞检测与得分统计

6.背景音乐：成背景音乐与音效系统

四、主要流程

1. 整体流程



2. 功能模块实现

（1）游戏主循环控制：

通过 while(running)主循环实现 144FPS 固定帧率控制，采用 GetTickCount()计算时间差与 Sleep()补偿机制保证帧稳定性。该模块包含输入处理、状态更新、渲染三阶段循环流程，符合游戏主循环的经典设计模式。

（2）角色控制系统

玩家控制：

Player 类实现键盘事件处理（ProcessEvent）与八向移动（Move 方法）。采用归一化向量计算移动方向，确保斜向移动速度恒定：

敌人 AI：

Enemy 类实现自动寻路算法，通过计算玩家位置的方向向量实现追逐逻辑。包含智能生成策略（地图边界随机出生）和碰撞响应系统。

（3）动画系统

Animation 类实现帧动画管理，关键特性包括：

时间驱动帧切换（interval_ms 控制播放速度）

资源自动释放（析构函数释放 IMAGE 对象）

支持多方向动画切换（左右移动/静止面朝向）

-

（4）用户界面模块

场景管理：

通过 Scene 枚举实现菜单与游戏场景切换，采用双缓冲绘图技术（BeginBatchDraw/EndBatchDraw）消除画面闪烁。

按钮系统：

Button 抽象类及其子类实现：

三态按钮效果（空闲/悬停/按下）

鼠标点击区域检测（CheckCursorHit）

事件回调机制（OnClick 虚函数）

（5）物理与碰撞系统

碰撞检测：

采用轴对齐包围盒(AABB)算法。

伤害计算：

实现 3 发子弹命中判定机制（hit_bullets[3]数组跟踪），支持多段伤害累计。

（6）资源管理系统

图像资源池：

使用 IMAGE 数组预加载玩家/敌人动画序列（6 帧玩家动画、5 帧敌人动画）

音频管理：

通过 mciSendString 实现背景音乐与音效的异步播放，支持多音轨控制。

（7）战斗系统

子弹轨迹：

采用极坐标参数方程生成旋转弹道：

计分机制:

通过 DrawPlayerScore 实现实时分数渲染，采用透明背景文字绘制技术.。

(8)对象生命周期管理

敌人生成:

TryGenerateEnemy 实现动态难度提升，初始 80 帧生成间隔，每生成一个敌人缩短 2 帧间隔（最低 20 帧）

内存管理:

使用 vector<Enemy*>动态容器，通过 swap-pop_back 模式实现高效对象回收。

五、核心技术与算法

1、核心模块技术体系

动画控制系统：

时间驱动模型：采用增量时间累加机制，实现帧率无关的动画播放。

多状态切换：玩家角色根据移动方向自动切换左右动画序列，静止时使用独立面朝动画。

透明混合绘制：通过 AlphaBlend 函数实现带 Alpha 通道的贴图渲染。

碰撞检测系统：

AABB 算法：基于坐标范围的快速相交检测，用于玩家-敌人、子弹-敌人的初步碰撞筛选。

多段命中机制：通过 hit_bullets[3]数组记录子弹命中状态，从而实现多次伤害累计。

精确坐标映射：敌人碰撞点计算时采用中心点坐标偏移，提升检测精度。

物理运动系统：

刚体动力学：速度向量分解实现八方向平滑移动。

运动约束：通过边界检测防止角色移出屏幕。

动态难度调整：敌人生成间隔从 80 帧逐步缩短至 20 帧，提升挑战性。

2、关键算法实现

AI 追踪算法：

向量追踪法 ：实时计算玩家与敌人的方向向量，实现基础追逐

行为状态机 ：通过 facing_left 布尔值管理敌人朝向状态

子弹轨迹算法：

极坐标方程：通过正弦函数叠加产生径向波动效果

切向运动控制：基于系统时间戳的相位差生成旋转轨迹

场景管理算法:

状态模式：通过枚举类型管理菜单/游戏场景切换

双缓冲绘制：使用 BeginBatchDraw/EndBatchDraw 消除画面撕裂

时间控制算法：

主动休眠补偿：通过 Sleep 函数维持稳定帧率

增量时间传递：将 delta_t 参数传递给动画系统实现时间同步

六、遇到的问题与解决方案

1. 问题：角色图像带有黑色底边，透明度缺失

分析：EasyX 默认的 putimage 函数不支持 Alpha 透明通道的直接渲染。当加载带有透明背景的 PNG 图片时，透明像素会被默认为黑色（RGB 0x000000），从而导致黑色底边。

解决方案：使用自定义 Alpha 混合函数，通过逐像素计算 Alpha 通道值，手动实现透明混合。

2. 问题：角色移动时卡顿且顿挫感严重

分析：KYE_DOWN 消息在按键按下的一段时间后才会接连被触发，KEY_DOWN 消息的产生与主循环异步进行

解决方案：在 Player::ProcessEvent 中通过布尔标志记录按键状态变化，通过 is_move_left 等状态标志，将离散的 KEY_DOWN 事件转化为连续状态，避免依赖单个 KEY_DOWN 事件的时序，直接通过标志位判断当前移动方向。在 Player::Move 中结合增量时间（Delta Time）实现平滑移动，通过 delta_time 参数实现帧率无关的动画更新。

七、测试

1. 单元测试(模拟):

在开发过程中对关键类和函数进行了隔离测试和日志输出验证，例如:

测试能否正确添加和移除敌人。

测试 player 的移动是否符合预期。

测试碰撞检测算法的准确性。

集成测试:

运行整个游戏，测试各模块组合后的功能:

游戏流程是否顺畅(开始、游戏、升级、结束)。

UI 元素是否按预期显示和交互。

背景音乐和音效是否播放。

八、收获与总结

1. 面向对象编程的深化理解

（1）封装的价值

将 Player 的移动状态设为私有，避免外部直接修改导致的逻辑混乱，通过 GetPosition()方法控制坐标访问，确保碰撞检测的正确性。

动画系统封装帧计算细节，主程序只需调用 Play()接口。

（2）继承的优势

大大减少按钮系统代码量，降低新增功能开发时间，通过基类统一处理鼠标事件，子类专注业务逻辑。未来扩展设置按钮、存档按钮时只需新增子类。

2. 工程实践能力提升

掌握 CMake 跨平台编译配置

理解资源管理的重要性（图片、音频的统一加载）

学会使用版本控制工具（Git）进行迭代开发

3. 性能优化意识

通过对象池技术减少 Enemy 的频繁创建/销毁

使用批量绘图(BeginBatchDraw)提升渲染效率

九、收获与总结

1.项目概述

本项目基于 C++与 EasyX 图形库，开发了一款“幸存者”类游戏，实现了角色操控、敌人生成与追踪、动态子弹轨迹、多场景切换（菜单/游戏）、碰撞检测、得分统计等核心功能。项目通过事件驱动架构和面向对象设计，结合动画系统、资源管理模块及音效交互，展现了完整的游戏开发流程与实时交互逻辑。

2.技术实现亮点

（1）图形渲染与动画系统

使用 AlphaBlend 实现透明贴图渲染（如角色阴影与子弹特效），避免黑边问题

基于时间戳的动画帧管理（Animation::Play），支持角色多方向动画切换（移动/静止），帧间隔通过 delta_time 实现帧率无关的平滑过渡

极坐标驱动的子弹环绕特效（UpDateBullets），结合正弦函数动态调整半径与相位差，提升视觉表现力

（2）事件处理与游戏逻辑

利用 peekmessage 实现非阻塞式消息轮询，分离输入事件处理与主循环逻辑

通过状态机管理玩家移动方向（Player::ProcessEvent），结合向量归一化消除按键抖动，确保角色平滑移动

动态敌人生成系统（TryGenerateEnemy），通过计数器逐步缩短生成间隔，提升游戏难度

（3）资源管理与性能优化

预加载图像资源（loadimage）与音效文件（mciSendString），避免运行时卡顿

双缓冲绘图（BeginBatchDraw/EndBatchDraw）减少画面撕裂，结合增量时间（Delta Time）稳定帧率

对象池技术动态管理敌人实例（vector<Enemy*>），降低内存碎片化风险

3. 核心收获

（1）C++面向对象设计能力

封装 Player、Enemy、Bullet 等实体类，通过继承与多态实现差异化行为（如敌人追踪逻辑、子弹轨迹计算）

掌握资源生命周期管理，正确使用 delete 释放动态内存，避免内存泄漏

（2）事件驱动架构实践

理解消息队列机制的异步特性，通过 WM_KEYDOWN 和 WM_KEYUP 事件实现精准输入响应

设计多场景切换逻辑（Scene::MENU 与 Scene::GAME），初步实践 UI 与逻辑分离思想

（3）调试与性能优化经验

通过日志分析与断点调试解决音效播放异常（如 mciSendString 参数错误）和碰撞检测失效问题

优化碰撞检测算法（基于 AABB 包围盒），平衡性能与精度需求

4.总结

本项目从零构建了一个功能完整的游戏原型，深化了对 C++面向对象编程、

图形渲染原理和实时交互逻辑的理解。通过解决事件异步处理、资源生命周期管理、跨平台兼容性等复杂问题，系统性提升了工程化思维与调试能力。尽管存在碰撞检测精度、AI 行为单一等不足，但为后续开发复杂游戏奠定了坚实基础。未来将聚焦性能优化与架构升级，探索更高效的开发模式与技术方案。