高级语言程序设计
实验报告

南开大学 工科实验班大类
姓名：张惟一
学号：2311917
班级：工科实验班 3-1 班
2024 年 5 月 16 日

目录

高级语言程序设计大作业实验报告

一. 作业题目

不基于既有 GUI 框架（Qt、MFC、EasyX）的游戏开发。

二. 开发软件

Visual Studio Code、CMake、MinGW 13.2.0

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

- 框架层

  此层以 game_window 类为核心，管辖以下 4 个“经理”类：
  - render_manager：管理基础渲染功能
  - texture_manager：管理纹理加载，使用 LRU 缓存减少重复加载
  - input_manager：管理键盘输入和键位映射
  - activity_manager：管理“活动类”

  其中 activity_manager 实现了两大核心机制：消息队列与活动栈。
  - 消息队列：每个消息是一个 std::function 对象，它被放入一个线程安全的环形队列。同时，有一个线程不断从这个队列中取出消息并执行。这种机制保证了消息会按入队顺序在单线程中顺序执行。
  - 活动栈：一个存放“活动”基类指针的 std::vector。程序启动时，活动栈中会压入一个初始活动。当活动栈为空时，程序结束。

    通过消息队列，可以对活动栈进行下列操作：
    - “下一个”：删除栈顶的活动，用新的活动代替，栈大小不变

◆ “调用”：向栈中压入一个新的活动，栈大小+1

◆ “退出”：删除并弹出栈顶的活动，栈大小-1

由此可以实现像调用函数一样“调用”其他活动的效果。

注：活动基类的代码如下，利用了虚函数的动态特性。

```
struct iface_activity {

    virtual ~iface_activity() {};

    virtual void render() = 0;

    virtual bool vkey_only() { return true; }

    // 仅在键盘状态真正改变时触发

    virtual void on_key_change() {}

    // 在有键盘事件时触发，例如按住某键时会持续触发

    virtual void on_key_signal() {}

    virtual void on_tick(double this_time, double last_time) {}

};
```

- 游戏业务层

  此层包含具体游戏逻辑的实现，目前有以下几个类：

  ■ acts::avg_scene：目前大多数场景的基础

  ■ acts::black_jack_scene：21 点游戏的实现

  ■ acts::plane_battle_scene：飞机大战游戏的实现

  其中 acts::avg_scene 并不包含具体场景的内容，它根据构造场景时传入的 name 参数在静态成员 scene_scripts 中查找对应的场景脚本，由指定的脚本填充场景的具体内容。

  除此之外，简单物理模拟器算法也暂时位于外层，简单物理模拟器由 naive_engine::simulator 类实现，其中使用了松散网格四叉树来高效检测碰撞。

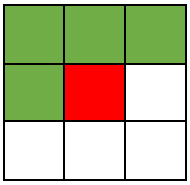
2. 算法或公式

松散网格四叉树是碰撞检测的核心算法，它可以有效地缩小每个元素碰撞检测的范围，从而降低碰撞检测所需的时间。同时，节点出口范围大于入口范围的特性

使它在实体不断运动的情况下也能保持较高的效率。
本程序采用节点出口范围大小为入口范围 2 倍的四叉树，这种情况下，只要实体的中心处于入口范围中，那么它整体一定处于出口范围中。
本程序实现时使用一个双向映射（boost::bimap）代替了真正的树，以保证实体在树中唯一。映射的每一对为(实体下标,实体指针)，左侧为 unordered_multimap，右侧为 unordered_map。

对松散网格四叉树定义如下概念与操作：

- “实体下标”：整个场景分为多层网格，第 1 层只有一个包含整个场景的，每增加 1 层，每个网格长和宽减半。一个实体属于网格大小大于其大小的层中层数最大的那层中包含它中心的网格。实体的下标等于它属于的网格在该层中的下标加上所有更低层的网格数。（每层的网格数为前一层的 4 倍，更低层的网格数可以用等比数列公式 O(1)求出）
- “插入实体”：计算实体下标并向双向映射中插入一对数据。
- “移除实体”：在双向映射的右侧查找实体指针，删除对应数据。
- “移动实体”：将实体的位置加上一个向量，然后计算新的实体下标。如果下标改变，那么在双向映射的右侧查找实体指针，修改对应数据。
- “对应的低层网格”：对于一个网格，在更低层，每层中完全包容它的网格只有一个，称该网格为对应的低层网格。
- “向低层搜索”：搜索一个网格的所有“对应的低层网格”及以这些低层网格为中心的九宫格。
- “在本层搜索”：搜索一个网格同层的左上四宫格，如图：（注：在本程序的实现中，upsearch 函数完成了“向低层搜索”和“在本层搜索”）



- “碰撞检测”：在 unordered_multimap 中，不同元素之间的顺序是不确定的，但相同元素总是相邻的。对每一个实体先“向底层搜索”，再“在本层搜索”，最后在每一组下标相同的实体内部相互检测，即可保证不重不漏。

3. 单元测试

1. 基本测试：测试游戏是否能正确切换场景、执行游戏逻辑
2. 资源占用测试：通过任务管理器观察测试游戏的 CPU 与内存占用

五. 单元测试

测试结果

基本测试：通过
资源占用测试：CPU 占用维持在 10%以内，内存维持在 1G 以内，无内存泄漏

六. 收获

1. 了解理论知识：该代码涉及到诸多底层概念，如消息队列、调用栈与 OpenGL 着色器，丰富了我的知识。

2. 深入理解算法：该代码中涉及的算法包括松散网格四叉树和 LRU 缓存，编写这些算法的实现增加了我的编码经验。

3. 提高调试能力：编写该代码的过程中程序不正常运行的情况是常有的，但最终都被我以二分调试、gdb 断点、采集热点函数等方式解决。调试与修复的过程增强了我逻辑思考与搜集信息的能力。