OOP 大作业文档

背景介绍

目前网络上有各种各样的图形库，游戏开发者可以用于开发游戏，但这些图形库多使用不便，并且从库中底层的接口开始构建游戏引擎更加繁琐。我们的大作业实现了这样的游戏引擎，使开发者不必考虑游戏的架构，直接编写游戏逻辑即可。为了实现多平台上的游戏开发，我们实现了两个功能相似的游戏引擎，一个使用 OpenGL 图形库，一个使用 DirectX 图形库。这两个版本分别附带一个示例小游戏。本文档介绍用 DirectX 实现的版本。

Q：OpenGL 是跨平台的，为什么需要 DirectX 版本？

A：经过实验，OpenGL 在 Windows 上的运行效果较差，而目前主流的在 PC 上玩游戏使用的操作系统是 Windows，因此我们认为编写一个在 Windows 专用的版本很有必要，我们决定用 DirectX 实现此版本。

需求分析

- 渲染目标，图像工厂，文字工厂，消息循环及消息处理等底层功能，可封装在一个主类中。
- 用于显示文字或图片的 Object 类，一个 Object 可能具有一定移动方式和产生其他 Object 的方式，可通过成员变量函数指针实现。
- 用于保存关卡信息的 Stage 类，一个 Stage 保存关卡进行时间，

Object 的出现情况等信息，并且提供由关卡当前帧的状态过渡到下一帧的状态的接口（Stage 类的设计因游戏而异，在示例游戏中提供的仅是开发射击游戏时的一种实现方法）

分工

赵鋆峰：游戏逻辑；组件设计等

郑少锟：系统架构；图形引擎（OpenGL）；资源管理等

孙志航：图形引擎（DirectX）；人体工程学设备等

王展鹏：物理引擎；数学引擎等

阎婧雅：游戏设计；音频引擎；网络通信等

框架设计

- 主类 DemoApp 实现底层的渲染目标、图片文字工厂等资源的创建以及绘图类方法，并用一个 vector 变量 Bitmaps 保存正在被使用的位图信息，另一个 vector 变量 Brushes 保存正在被使用的笔刷信息。
- BitmapObject 类保存图像的尺寸和中心位置，图像本身保存在 DemoApp 中，BitmapObject 有一个指向 DemoApp 的 Bitmaps 的“指针”（实际是 vector 中元素的位置）。
- BitmapObject 有多个派生类，包括可匀速滚动的 SceneBitmapObject，可平移旋转的 TempBitmapObject，有生命、攻击并且可以生成其他 Object 的 CreatureObject。（根据游

戏的不同，开发者可以设计 BitmapObject 的新派生类实现更多功能）

- Stage 类保存关卡状态，根据游戏的不同类型有多种设计，但主要结构均为若干个 BitmapObject（或其派生类）的 vector 成员，Process 类方法用于将上一帧的状态转换为下一帧的状态，Reset 类方法用于重置关卡。
- 消息循环作为 DemoApp 的类方法，用 Windows 消息循环实现，每隔 33 毫秒向消息循环发送一条绘制信息，这样可以令游戏以 30 帧每秒运行。
- 绘制函数作为 DemoApp 的类方法，根据目前的关卡信息，调用对应 Stage 对象的 Process 函数。

技术细节

开源资源

DemoApp 类的定义部分来自 MSDN 的 Direct2D 教程（https://msdn.microsoft.com/en-us/library/windows/desktop/dd370994(v=vs.85).aspx），其余为自行完成。

编程技巧

- DemoApp 在程序初始化时创建且仅创建一个，用到了单例模式。
- BitmapObject 及其派生类与 TextObject 均有一个指向 DemoApp 中的 vector 中的元素的指针，用到了桥接模式。

- CreatureObject 中有 Bullet 成员，调用 CreatureObject 的 Fire 类方法时，首先用 CreatureObject 中的 ShouldFire 类方法（实际为函数指针）判断是否达到开火条件，若是，则调用 Bullet 成员的 Fire 类方法，用到了代理模式。
- 为了保证消息处理函数的简洁，不需要为了增加操作而修改消息处理函数。键盘事件不用消息处理函数来处理，而是将开发者希望用键盘控制的 TempCreatureObject 的函数指针 Move 或 CreatureObject（由 TempCreatureObject 派生）的函数指针 ShouldFire 赋值为这样一个函数：它通过 Windows API 函数 GetKeyState 读取某个键是否被按下，并对调用它的 Object 做出相应的操作。

总结

在本次大作业的编写过程中，我充分了解了 OOP 的许多原则并不是纸上谈兵，而是会确实地降低代码的编写、维护、运行难度的。尤其是像我们的作品这样本身就是一个未完成品，要交由其他开发者完成的代码（本次作业提交了完成品，但只是一个示例），更需要考虑代码的维护成本，尤其是其可拓展性。另一方面，我知道了跨平台的并不是最好的，也要考虑它在每种平台上的运行效果，尤其是使用者主要使用的平台上的运行效果（如游戏主要在 Windows 运行）。