高级程序设计 课程设计报告

151220091 沈明杰

1 课程设计的内容和目标

使用 C++语言和微软的 MFC 类库 ,设计并实现一个带有 GUI 界面的飞机游戏。游戏包含玩家飞机、敌机、子弹等元素，游戏者可以操纵玩家飞机移动，通过发射子弹击毁敌机。

1.1 游戏设计

玩家可以使用方向键（或 ADWS 键）控制飞机移动，按空格键发射子弹。玩家通过发射子弹击中敌机来获得分数。每隔一段时间，会有一架新的敌机出现。敌机一共有三种类型，每种类型的敌机的血量不同：第一种敌机（普通敌机）只需一枚子弹即可击毁，第二种（中级敌机）需要两枚，而第三种（Boss 敌机）需要 10 枚子弹才可击毁。每类敌机出现的频率不同，第一类出现的频率最高，第二类其次，而血量最大的第三类敌机出现的频率最低。玩家在游戏时需要躲避敌机，当任意一架敌机与玩家飞机发生碰撞后，则游戏结束。

1.2 课程设计目标

综合运用本学期高级程序课程中学到知识：

- 面向对象程序设计，数据抽象和封装（类/对象），继承（基类与派生类、子类型）以及消息的多态和动态绑定（虚函数）；
- 泛型（类属）程序设计，基于 STL（标准模板库）容器、迭代器和算法的编程等；
- 事件（消息）驱动程序设计，包括：消息、消息循环以及消息处理过程等；
  基于“文档 - 视”结构的应用框架的程序设计，包括：文档-视应用框架，MFC 类库，windows 应用程序设计等。

设计良好的类层次结构，尽量使代码易读，易维护。

2 开发环境

编程语言：C++（C++11）

IDE： Visual Studio 2015

框架：MFC

OS: Windows

3 类层次关系和实现

类层次结构设计是面向对象程序设计的最重要部分。本程序的类层次关系如下图所示：

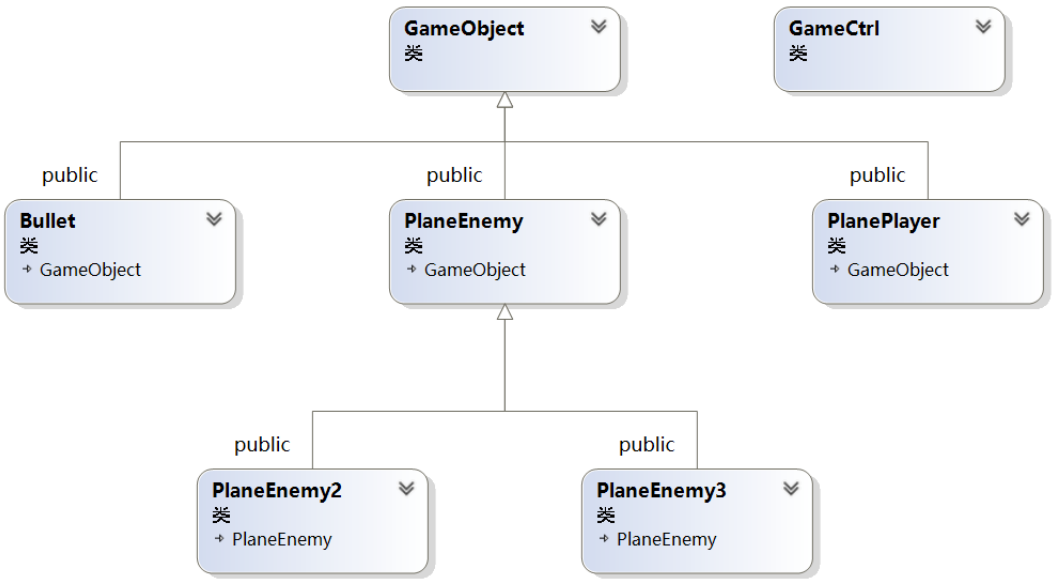

图 1 类层次结构图

GameObject 类是一个抽象基类，描述对所有游戏元素公共的属性和方法，它拥有三个直接派生类：Bullet 类，PlaneEnemy 类和 PlanePlayer 类。Bullet 类和 PlanePlayer 类分别描述子弹和玩家飞机的属性和方法。PlaneEnemy 类描述普通敌机，它拥有两个派生类：PlaneEnemy2 类和 PlaneEnemy3 类，分别描述中级敌机和 Boss 敌机。

此外，我们还定义了一个 GameCtrl 类，用于管理所有游戏元素。

3.1 GameObject 类

GameObject 类是玩家类（PlanePlayer），敌机类（PlaneEnemy）和子弹类（Bullet）这三个类的基类，它描述了玩家、敌机和子弹所共有的属性和方法。GameObject 类有纯虚函数，因此它是一个抽象基类。

数据：

名称	类型	访问控制	描述

pos	CPoint	protected	位置（横纵坐标）
height	int	protected	高度
width	int	protected	宽度
dir	Dir	protected	移动方向（Dir 是一个枚举类型，它有四个枚举符 LEFT, RIGHT, UP, DOWN）
speed	int	protected	速度，每次移动所改变的像素值

方法：

名称	类型	访问控制	描述
GameObeject	无	protected	构造函数，初始化各数据成员
getX	int	public	访问函数
getY	int	public	访问函数
getHeight	int	public	访问函数
getWidth	int	public	访问函数
getImage	const CImage &	public	纯虚函数，返回游戏元素对应的图片
move	bool	public	用于游戏元素位置的移动

我们需要对 move 和 getImage 这两个成员函数做进一步解释。

3.1.1 GameObject::move

move 成员函数的原型是 bool GameObject::move(int cliHeight, int cliWidth)。它接受两个 int 型参数，分别表示客户区的高度和宽度。函数返回值是 bool 类型，如果游戏对象移动后仍然位于客户区内部，则返回 true，否则返回 false。

具体实现的时候，我们根据表示运动方向的 dir 成员，使用一个 switch 语句。在每个 switch 语句的分支中，判断移动后是否会超出边界，如果不会超出，则向 dir 给出的方向移动 speed 个单位，并且返回 true；否则，返回 false.

之所以加入 dir 数据成员来表示移动方向，而不是将运动方向作为 move 函数的一个参数的原因是：子弹和敌机的移动方向都是固定的，子弹一定向上运动，敌机一定向下运动。如果每次调用子弹或敌机的移动函数时，都要额外再传一个位置信息，则稍显累赘。

将对象移动这一功能放在基类实现，可以使派生类直接继承该方法。我们不必为子弹、玩家、敌机分别编写 move 函数，从而有效避免了代码的冗余。

3.1.2 getImage 成员函数

该成员的声明是 virtual const CImage &getImage() const = 0;这是一个纯虚函数。

这一函数的设计需要一定的解释。我们知道，每类敌机和子弹都有对应的图片，我们想让每个类，而不是类的对象，存储与之对应的图片，因为每个类的对象都共享同一张图片。以子弹类为例，所有子弹类的对象都共享同一张子弹图片，因此我们没有理由将这张图片存储在每个子弹类的对象中，这样做空间效率太低。在 C++中，我们可以将存储图片的 CImage 对象声明为 static。

不幸的是，这么做又带来新的问题。既然玩家类，子弹类，敌机类都需要一个静态的 CImage 成员，很自然的想法是，将这个成员放到基类中，使它的派生类能够继承这一变量。但是，C++语言规定，静态变量在整个类层次中只有一个实例，不管基类拥有多少子类，每个静态成员有且仅有一个实例：

If a base class defines a static member (§7.6, p. 300), there is only one such member defined for the entire hierarchy. Regardless of the number of classes derived from a base class, there exists a single instance of each static member.

——C++ Primer 5th Edition, Chapter 15

因此，我们不得不在玩家类，子弹类，和每个敌机类中都定义一个静态的 CImage 对象，注意，这些静态变量的名字是可以相同的，但它们是不同的变量。

为了向外界提供统一的访问图片的接口，也为了得到运行时的动态绑定效果，我们在 GameObject 类中定义了 getImage 函数，用于封装对静态变量的访问。将其定义为虚函数的原因是：每个派生类需要重写这一函数的实现；将其定义为纯虚函数的原因是：GameObject 类根本不包含静态的 CImage 对象，自然无法实现这一函数，将 getImage 函数定义在基类（GameObject 类）是为了向外界提供统一的访问接口。

函数返回值类型是 CImage 对象的常量引用，返回引用的原因是为了提高效率，避免不必要的拷贝操作，返回常量引用的原因是，GameObject 类不希望外部通过这个引用改变自己的 CImage 数据成员。

3.2 PlanePlayer 类

PlanePlayer 类继承 GameObject 类，描述了玩家飞机的属性和方法

数据：

名称	类型	访问控制	描述
image	CImage	private	存储玩家飞机的图片

图 2 玩家飞机

方法：

名称	类型	访问控制	描述

PlanePlayer	无	public	构造函数，初始化各数据成员
setDir	void	public	设置表示运动方向的 dir 变量
getImage	const CImage &	public	重写父类的虚函数，返回 image 对象的常量引用

这里的 image 对象并不是静态成员，考虑到今后我们只会创建一个 PlanePlayer 对象，不声明成 static 也不会造成空间的浪费。在构造函数中，通过调用基类的构造函数初始化属于基类部分的成员，然后执行 image 的 Load 方法，加载玩家飞机的图片。

3.3 Bullet 类

Bullet 类继承 GameObject 类，描述了子弹的属性和方法。

数据：

名称	类型	访问控制	描述
image	CImage	private	static 成员，存储子弹的图片

图 3 子弹

方法：

名称	类型	访问控制	描述
Bullet	无	public	构造函数，初始化各数据成员
initImage	void	public	static 成员，用于加载图片
getImage	const CImage &	public	重写父类的虚函数，返回 image 对象的常量引用

3.4 PlaneEnemy 类

PlaneEnemy 类继承 GameObject 类，描述了敌机（普通敌机）的属性和方法。

数据：

名称	类型	访问控制	描述
image	CImage	private	static 成员，存储普通敌机的图片

health	int	private	敌机的血量

图 4 普通敌机

方法：

名称	类型	访问控制	描述
PlaneEnemy	无	public	构造函数，初始化各数据成员
initImage	void	public	static 成员，用于加载图片
getImage	const CImage &	public	重写父类的虚函数，返回 image 对象的常量引用
decreaseHP	bool	public	血量减一。如果减去一滴血后 ,血量仍大于 0，则返回 true，否则返回 false

3.5 PlaneEnemy2 类

PlaneEnemy2 类继承 PlaneEnemy 类，描述了中级敌机的属性和方法。

数据：

名称	类型	访问控制	描述
image	CImage	private	static 成员，存储中级敌机的图片

图 5 中级敌机

方法：

名称	类型	访问控制	描述
PlaneEnemy2	无	public	构造函数，初始化各数据成员
initImage	void	public	static 成员，用于加载图片
getImage	const CImage &	public	重写父类的虚函数，返回 image 对象的常量

			引用

3.6 PlaneEnemy3 类

PlaneEnemy3 类继承 PlaneEnemy 类，描述了 Boss 敌机的属性和方法。本类与 PlaneEnemy2 类极为类似，惟一的区别是静态成员 image 所存储的图片不同。

数据：

名称	类型	访问控制	描述
image	CImage	private	static 成员，存储 Boss 敌机的图片

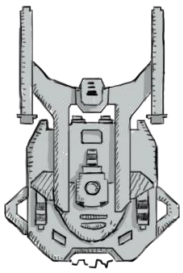

图 6 Boss 敌机

方法：

名称	类型	访问控制	描述
PlaneEnemy3	无	public	构造函数，初始化各数据成员
initImage	void	public	static 成员，用于加载图片
getImage	const CImage &	public	重写父类的虚函数，返回 image 对象的常量引用

3.7 GameCtrl 类

记录所有与游戏有关的数据，提供更新游戏逻辑、判断游戏是否结束的方法。

数据：

名称	类型	访问控制	描述
player	PlanePlayer	private	玩家飞机
blist	std::list<Bullet>	private	画面中所有子弹组成的列表

elist	std::list<std::shared_ptr<PlaneEnemy>>	private	画面中所有敌机组成的列表
score	int	private	玩家得分
height	int	private	客户区高度
width	int	private	客户区宽度

这里对elist成员做进一步说明。std::shared_ptr是C++11标准库提供的一种智能指针，它能提高使用动态内存（堆区）的程序的鲁棒性。智能指针的表现和普通指针类似，但是它会自动销毁它指向的对象，并且释放对象占用的内存空间。使用智能指针能使程序员不必显式使用new和delete操作，极大地简化了使用堆区的程序的编写。与普通指针类似，指向基类对象的智能指针也可以指向其派生类的对象。

使用std::list容器是为了高效地实现在任意位置的插入和删除操作。

方法：

名称	类型	访问控制	描述
GameCtrl	无	public	构造函数，初始化各数据成员
init	void	public	进一步初始化
playerMove	void	public	玩家移动
updateBullets	void	public	更新子弹的位置，若越界，则将其从 blist 移除
shoot	void	public	玩家飞机发射子弹，子弹从飞机头部发出
updateEnemies	void	public	更新敌机的位置，若越界，则将其从 elist 移除
addEnemy	void	public	在画面上部的某一位置随机生成一架敌机（生成 Boss 敌机的概率是 1/50，生成中级敌机的概率是 1/10，生成普通敌机的概率是 44/50）
bulletHitPlane	void	public	检测是否有子弹击中敌机，若发现某一子弹和某一敌机发生碰撞，则将该子弹从 blist 移除，并调用敌机的 decreaseHP 操作，如果敌机的血量减为 0，则将该敌机从 elist 中移除
gameOver	bool	public	判断游戏是否结束，若任意一架敌机与玩家飞机发生碰撞，则游戏结束
isConflict	bool	private	检测两个 GameObject 对象是否发生碰撞，该方法主要供 bulletHitPlane 和 gameOver 函数调用。该函数接受两个 const GameObject &类型的参数，体现了面向对象的多态特征。使用的算法是最简单的矩形碰撞检测算法。

还有一些访问函数，不具体列出，主要供 CPlaneView 类绘图使用。

3.8 CPlaneGameDoc类

拥有一个GameCtrl类型的数据成员，封装了读取和更新游戏数据的方法。

3.9 CPlaneGameView类

用于显示游戏数据，并与用户进行交互。下面简要说明几个重要的成员函数。

3.9.1 OnDraw

调用文档类的GetBkground()，GetPlanePlayer()，GetBulletList()，GetEnemyList()和GetScore()函数，在屏幕上绘制背景，玩家飞机，子弹，敌机和得分。

3.9.2 OnKeyDown

检测用户按下的键，调用文档类的PlayerMove(Dir)或Shoot()函数，实现玩家飞机的移动和发射子弹。

3.9.3 OnGameStart

游戏菜单下的开始按钮的处理程序，调用文档类的Init()函数，并设置两个定时器，定时器1的间隔为40ms，定时器2的间隔为500ms，两个定时器的功能见2.9.4.

3.9.4 OnTimer

定时器1：调用文档类的UpdateBullets()，UpdateEnemies()和BulletHitPlane()方法，更新游戏逻辑，判断游戏是否结束。

定时器2：调用文档类的AddEnemy()函数，增加一架敌机。

4 课程设计一和二的关联和衔接

课程设计一要求我们设计一个基于控制台的飞机游戏。基于控制台的飞机游戏和基于MFC的飞机游戏的区别主要在：

1） 游戏对象的输出。在控制台下，使用iostream标准输入输出类库的cout对象和Windows的API函数进行输出，每个游戏元素由若干字符组成。而在MFC下，我们通过CDC对象在屏幕上绘制游戏元素，每个游戏元素显示为一张图片。

2）游戏逻辑更新的方式。在控制台下，我们需要用一个主循环来控制游戏逻辑的更新，每次循环时处理键盘按键，更新游戏状态，绘制下一个画面。在MFC下，我们使用事件（消息）驱动的程序设计范式，在键盘按键事件（WM_KEYDOWN）处理程序中响应用户的输入，在定时器消息（WM_TIMER）处理程序中更新游戏状态，在绘图消息（WM_PAINT）处理程序中绘制下一个画面。

5 遇到的问题和思考

5.1 游戏画面闪烁

在游戏过程中，游戏画面出现严重的闪烁现象，影响用户体验。解决方法为添加WM_ERASEBKGND消息的处理程序，将

```
BOOL CPlaneGameView::OnEraseBkgnd(CDC* pDC)
{
	// TODO: 在此添加消息处理程序代码和/或调用默认值

	return CView::OnEraseBkgnd(pDC);
}
```

改为

```
BOOL CPlaneGameView::OnEraseBkgnd(CDC* pDC)
{
	// TODO: 在此添加消息处理程序代码和/或调用默认值

	return TRUE;
}
```

5.2 PNG图片的透明像素无法正确绘制

在MFC中使用CImage进行贴图的时候可以发现原本图片透明的地方也显示出了白色。解决方法是加载图片后，调用以下函数：

```
void make_transparent(CImage &img)
{
	for (int i = 0; i < img.GetWidth(); i++)
	{
		for (int j = 0; j < img.GetHeight(); j++)
		{
			unsigned char* pucColor = reinterpret_cast<unsigned char
*>(img.GetPixelAddress(i, j));
			pucColor[0] = (pucColor[0] * pucColor[3] + 127) / 255;
			pucColor[1] = (pucColor[1] * pucColor[3] + 127) / 255;
			pucColor[2] = (pucColor[2] * pucColor[3] + 127) / 255;
		}
	}
}
```

6 游戏效果展示

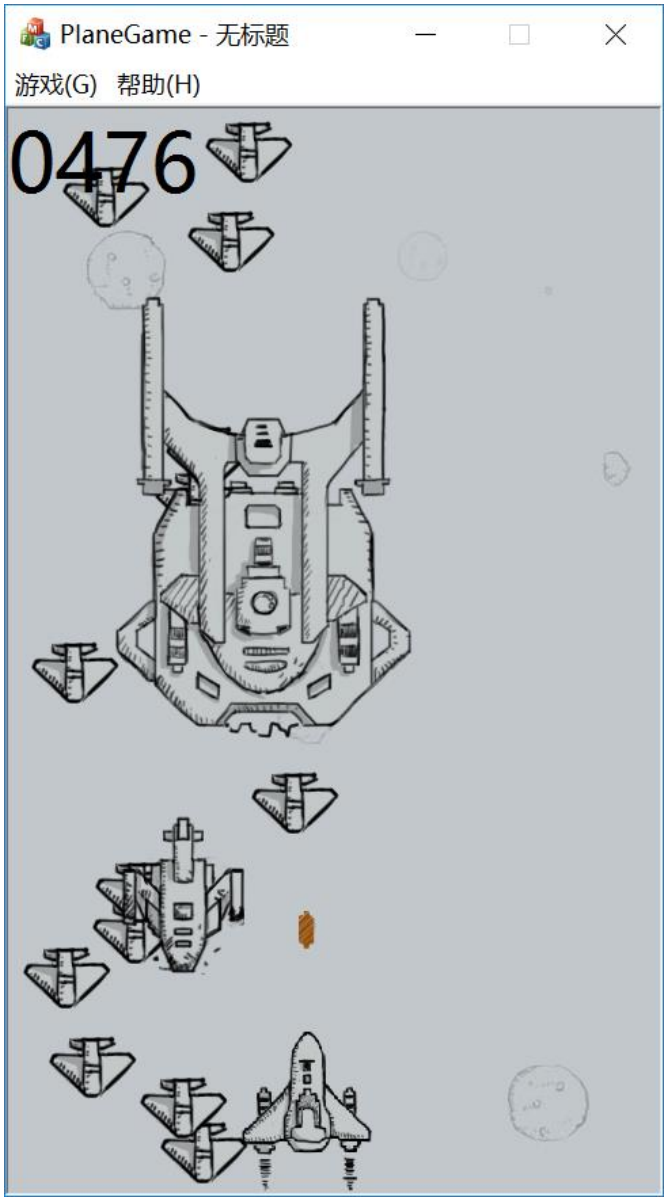

图 7 游戏画面展示

7 总结

与某些传统的课程设计（如图书管理系统）不同，本次课程设计的内容是设计并实现一个打飞机游戏，颇有趣味性，能激发我们的编程热情。通过本次课程设计，我们实践了理论课中学到的面向对象编程技术，进一步掌握了 C++语言，初步学会了基于 MFC 的 GUI 程序的开发。

参考资料

[1] Stanley B. Lippman, Josée Lajoie, Barbara E. Moo. *C++ Primer*, Fifth Edition. Addison-Wesley.

[2] http://blog.csdn.net/angelazy/article/details/10296341

[3] https://www.codeproject.com/Articles/33/Flicker-Free-Drawing-In-MFC

[4] http://www.cplusplus.com/forum/general/80589/