潍坊科技学院
Weifang University of Science and Technology

第11章 飞机大战

授课老师：刘国旭

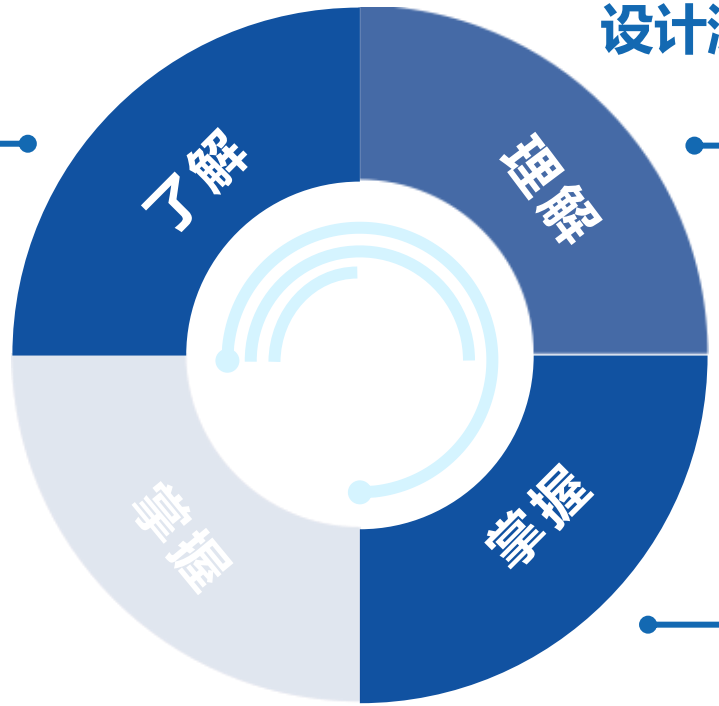
潍坊科技学院

- 游戏规则
- 面向对象思想
- pygame模块的使用

学习目标

1 了解 飞机大战游戏的规则

理解 面向对象思想，会独立设计游戏的类与模块 2



掌握 pygame模块的使用 3

目录页

目录页

目录页

11.1 游戏简介

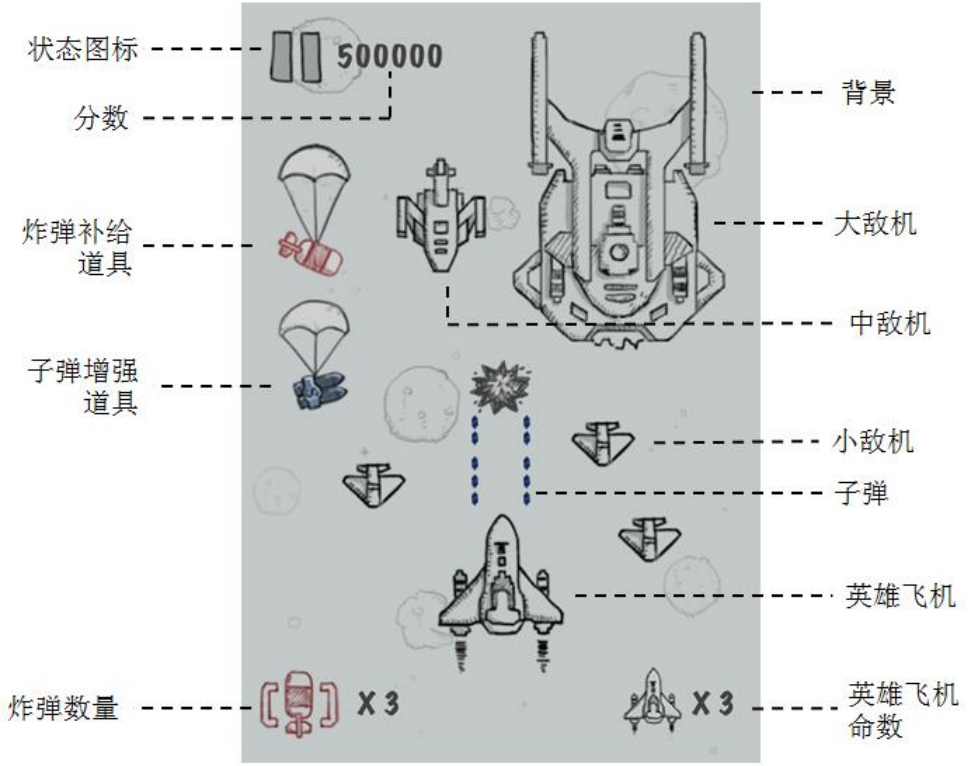
飞机大战是一款由腾讯公司微信团队推出的软件内置的小游戏，这款游戏画面简洁有趣，规则简单易懂，操作简便易上手，在移动应用兴起之初曾风靡一时。

11.1.1 游戏介绍

飞机大战游戏主要以太空主题的画面为游戏背景，由玩家通过键盘控制英雄飞机向敌机总部发动进攻，在进攻的过程中既可以让英雄飞机发射子弹或炮弹击毁敌机以赢取分数，也可以拾取道具以增强自己的战斗力，一旦被敌机撞毁就结束游戏。

11.1.1 游戏介绍

飞机大战游戏包含众多游戏元素，例如，大小各异的飞机、连发3颗的子弹、左上角的游戏分数等，其中主要的元素如图所示。



11.1.1 游戏介绍

飞机大战游戏中的主要元素可以归纳为背景、英雄飞机、敌机、道具、分数和奖励、关卡设定这几大类，各类中游戏元素的具体说明如下。

1.背景

整个游戏窗口的背景是一张星空图像，该背景图像会缓缓地向下持续移动，使玩家产生一种英雄飞机向上飞行的错觉。

11.1.1 游戏介绍

2.英雄飞机

英雄飞机是由玩家控制的飞机，是飞机大战游戏的主角，其相关说明具体如下。

（1）英雄飞机在游戏开始时显示在屏幕下方的中央位置。

（2）英雄飞机在出场后的3秒内处于无敌状态，此时它不会被任何敌机撞毁，也不会撞毁任何敌机。

（3）玩家可以通过方向键（↑、↓、←、→）控制英雄飞机在屏幕范围内向上方、下方、左方和右方移动。

2.英雄飞机

英雄飞机是由玩家控制的飞机，是飞机大战游戏的主角，其相关说明具体如下。

（4）英雄飞机出场后每隔0.2秒会自动连续发射3颗子弹，关于子弹的特征和行为具体如下：

- 特征：子弹的速度为12，杀伤力为1。
- 行为：子弹由英雄飞机头部的正上方位置发射，沿屏幕垂直方向向上飞行；子弹被发射时需要播放发射子弹的音效；子弹在飞行途中若击中敌机，会对敌机造成伤害；子弹若已飞出屏幕且飞行途中未击中任何敌机，会被销毁。

11.1.1 游戏介绍

2.英雄飞机

英雄飞机是由玩家控制的飞机，是飞机大战游戏的主角，其相关说明具体如下。

（5）英雄飞机出场后默认会携带3颗炸弹，玩家按下字母b时会引爆1枚炸弹，炸弹数量减1；炸弹数为0时，无法再使用炸弹。

（6）英雄飞机带有多个动画和音效。当飞机飞行时，显示飞行动画；当飞机被敌机撞毁时，显示被撞毁动画，并播放被撞毁音效；当飞机升级时，播放升级音效。

2.英雄飞机

英雄飞机是由玩家控制的飞机，是飞机大战游戏的主角，其相关说明具体如下。

（7）英雄飞机具有多条生命。英雄飞机的初始生命为3；英雄飞机被敌机击中时，命数减1；英雄飞机得分每增加10万，命数加1；英雄飞机的生命为0时，游戏结束。英雄飞机的命数会实时地显示在游戏界面的右下方位置。

11.1.1 游戏介绍

3.敌机

飞机大战游戏中敌机的机型小、中、大三种，各机型均有生命值、速度、分值、图片和音效等多个特征，且不同类型的敌机所具有的特征也不尽相同，后续在设计类时会有详细说明。

飞机大战中的敌机具有以下行为：

（1）敌机出现在游戏窗口顶部的随机位置。

（2）敌机按照各自不同的速度，沿垂直方向向游戏窗口的下方飞行。

（3）若敌机与英雄飞机相撞，则会击毁英雄飞机。

3.敌机

飞机大战中的敌机具有以下行为：

（4）若敌机被子弹击中，则敌机的生命值需要减去子弹的伤害度，此时根据敌机的生命值可以分如下两种情况进行处理：

- 如果敌机的生命值大于 0，那么显示敌机被击中图片（若有被击中图片），让敌机继续向屏幕下方飞行。
- 如果敌机的生命值等于0，那么播放敌机被撞毁动画与被撞毁音效。在被撞毁动画播放过程中，该敌机不会在屏幕上移动；在被撞毁动画播放完成后，该敌机被设为初始状态，跳转到第（1）步继续执行。

3.敌机

飞机大战中的敌机具有以下行为：

（5）若敌机飞出了游戏窗口且飞行途中没有被击毁，该敌机被设为初始状态。

值得一提的是，正在播放被撞毁动画的敌机是已经被摧毁的敌机，它既不能被子弹击中，也不能撞击英雄飞机。

11.1.1 游戏介绍

4.道具

在游戏过程中，道具每隔30秒会从游戏窗口上方的随机位置向下飞出，一旦飞出的过程中碰撞英雄飞机，就会被英雄飞机拾取。飞机大战游戏有两种道具：炸弹补给和子弹增强，关于这两种道具的功能描述如表所示。

道具名称	功能描述	速度	播放音效
炸弹补给	英雄飞机拾取后，炸弹数量加1	5	是
子弹增强	英雄飞机拾取后，发射的子弹由单排改为双排，且持续时长20秒	5	是

11.1.1 游戏介绍

5.分数和奖励

当英雄飞机通过子弹或炸弹击毁敌机时，会获得与敌机分值相对应的分数，并将获得的分数实时地显示在游戏窗口的左上方。同一局游戏的分数会不断累加，并在下一局开始时自动清零。

系统会记录玩家历次游戏所得到的最高分，并在游戏暂停和结束时显示在游戏窗口上。

11.1.1 游戏介绍

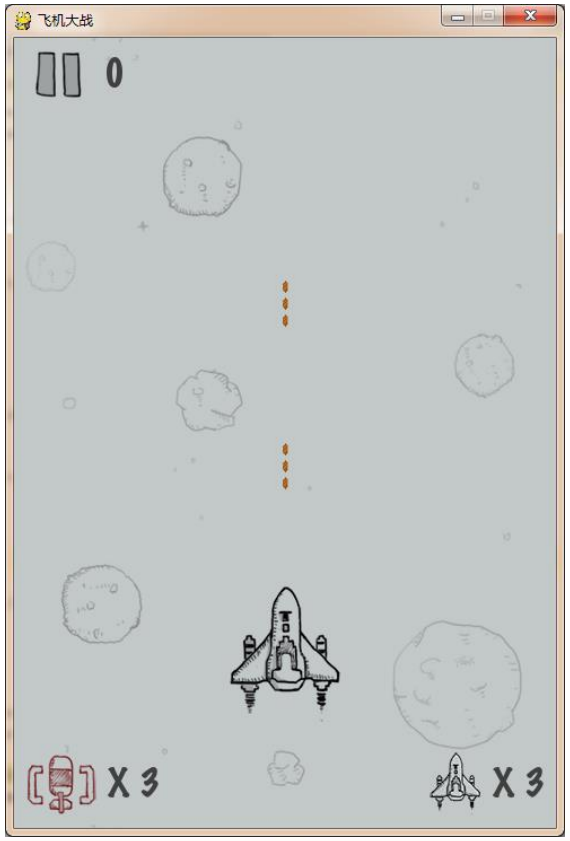
6.关卡设定

飞机大战游戏一共设立了3个关卡，依次是关卡1、关卡2和关卡3，其中关卡1为起始关卡。在英雄飞机的得分超过当前关卡的预设分值之后，游戏会自动进入下一个关卡。关卡越高难度越高，敌机数量和种类越多，速度也更快。3 个关卡的具体设定如表所示。

关卡名称	分值范围	小敌机数量（速度）	中敌机数量（速度）	大敌机数量（速度）
关卡 1	< 10000	16（1 ~ 3）	0（1）	0（1）
关卡 2	< 50000	24（1 ~ 5）	2（1）	0（1）
关卡 3	>= 50000	32（1 ~ 7）	4（1 ~ 3）	2（1）

11.1.2 游戏典型场景

飞机大战游戏的典型事件组成了各个典型场景，除了游戏进行中的场景之外，还包括游戏开始、飞机碰撞、游戏暂停、游戏结束几个场景。



目录页

11.2 项目准备

明确了飞机大战游戏之后，进入开发工作之前，我们需要先完成分析与设计等准备工作，以明确项目的实现方式和内部结构等。

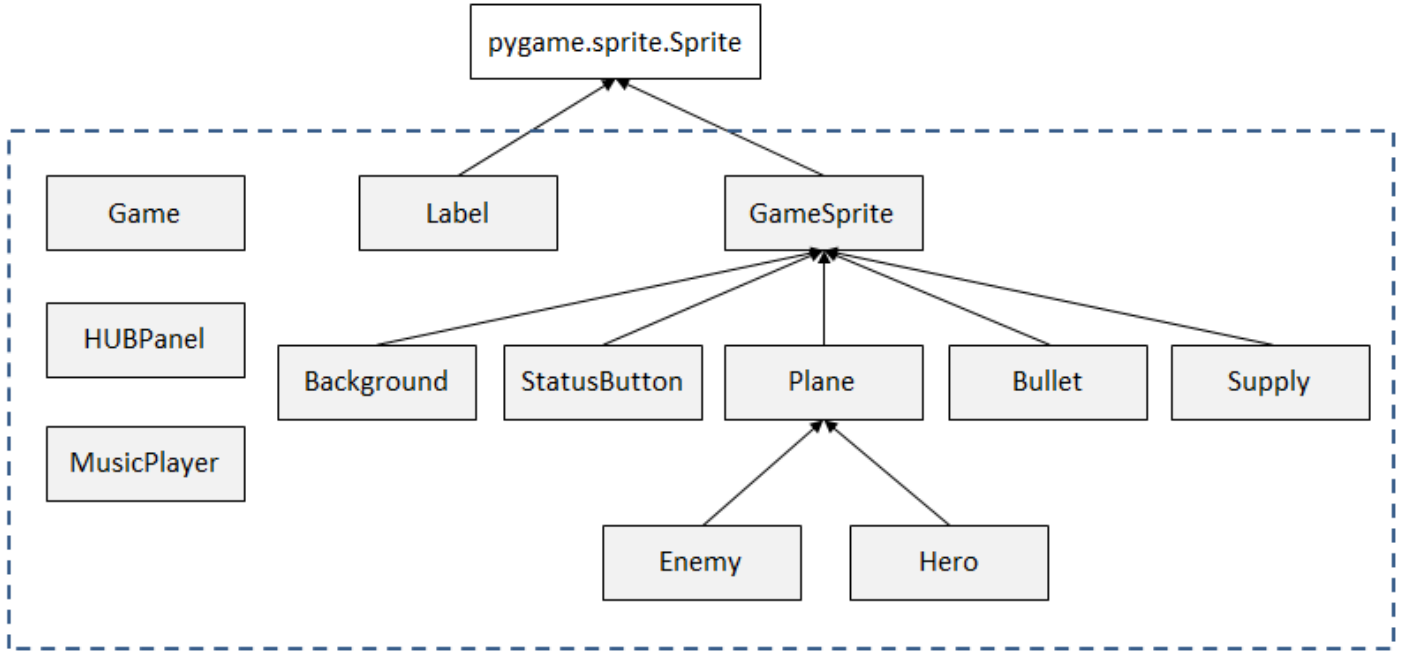
11.2.1 类设计

根据游戏介绍分析可知，飞机大战游戏有众多游戏元素，为方便统一管理游戏元素，我们按游戏元素的职责进行归类，各类的名称及说明如表所示。

分类	类名	说明
游戏类	Game	负责整个游戏的流程
指示器面板类	HUBPanel	负责统一管理游戏状态、游戏分数、炸弹数量、生命值以及文本提示等与游戏数据或状态相关的内容
音乐播放类	MusicPlayer	负责背景音乐和音效的播放
游戏背景类	Background	负责显示游戏的背景图像
状态按钮类	StatusButton	负责显示游戏的状态按钮
飞机类	Plane	表示游戏中的飞机，包括英雄飞机和敌机
子弹类	Bullet	表示英雄飞机发射的子弹
道具类	Supply	负责管理炸弹补给和子弹增强道具
文本标签类	Label	负责显示游戏窗口上的文本

11.2.1 类设计

飞机大战游戏项目中提炼出的类及类之间的继承关系如图所示。



11.2.2 模块设计

在设计程序时，我们既要考虑程序的设计理念和设计思想，又要考虑程序的结构。结构设计的原则是代码模块化、容易扩展和维护。因此，可将飞机大战游戏项目划分为4个模块：game.py、game_items.py、game_hud.py和game_music.py，各模块的说明如表所示。

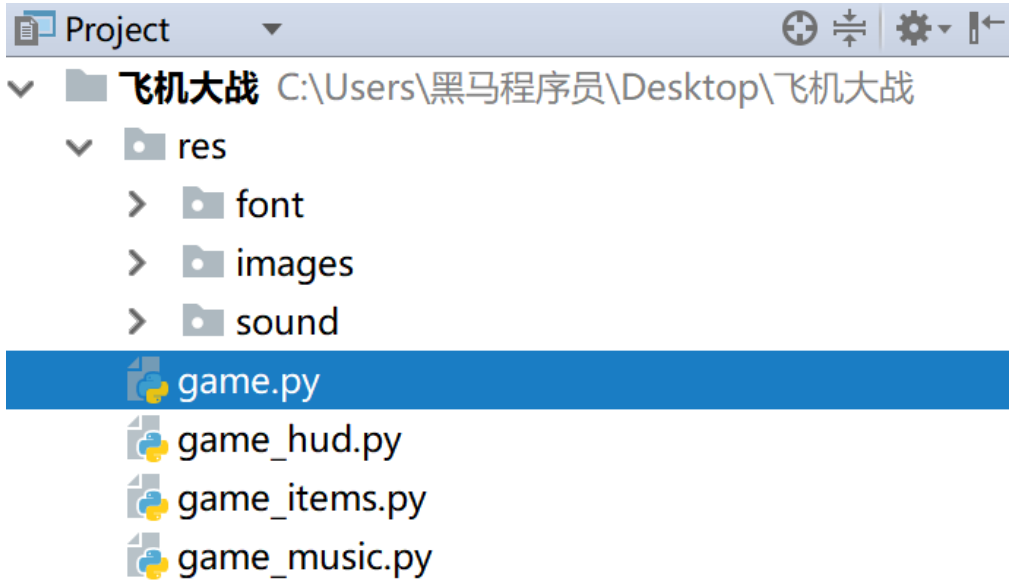
模块	说明
game.py	游戏主模块，封装Game类并负责启动游戏。
game_items.py	游戏元素模块，封装英雄飞机、子弹、敌机、道具等游戏元素类，并定义全局变量。
game_hud.py	游戏面板模块，封装指示器面板类。
game_music.py	游戏音乐模块，封装音乐播放器类。

11.2.3 创建项目

打开PyCharm工具，新建一个名称为“飞机大战”的项目。

在飞机大战项目中依次建立game.py、game_items.py、game_hud.py和game_music.py四个文件，并且将资源文件夹“res”复制到飞机大战目录下。

创建好的项目文件结构如图所示。

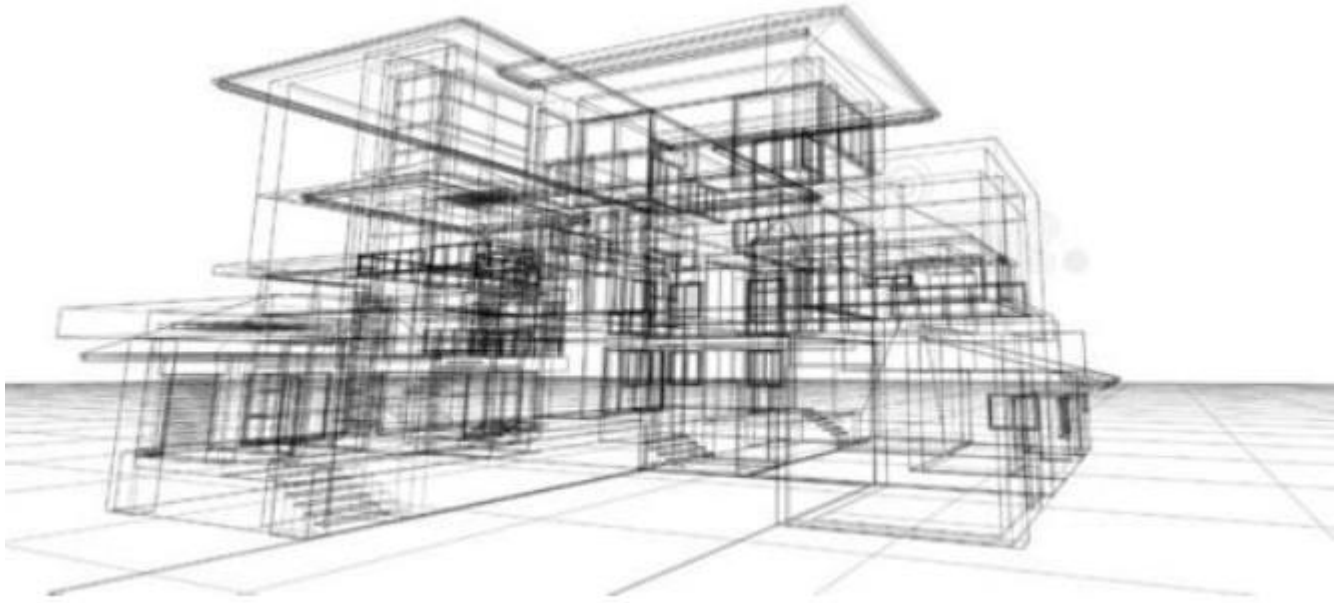


图中的res目录包含三个子目录：images、sound和font，子目录分别放置了游戏中使用的图片、声音和字体素材。

目录页

11.3 游戏框架搭建

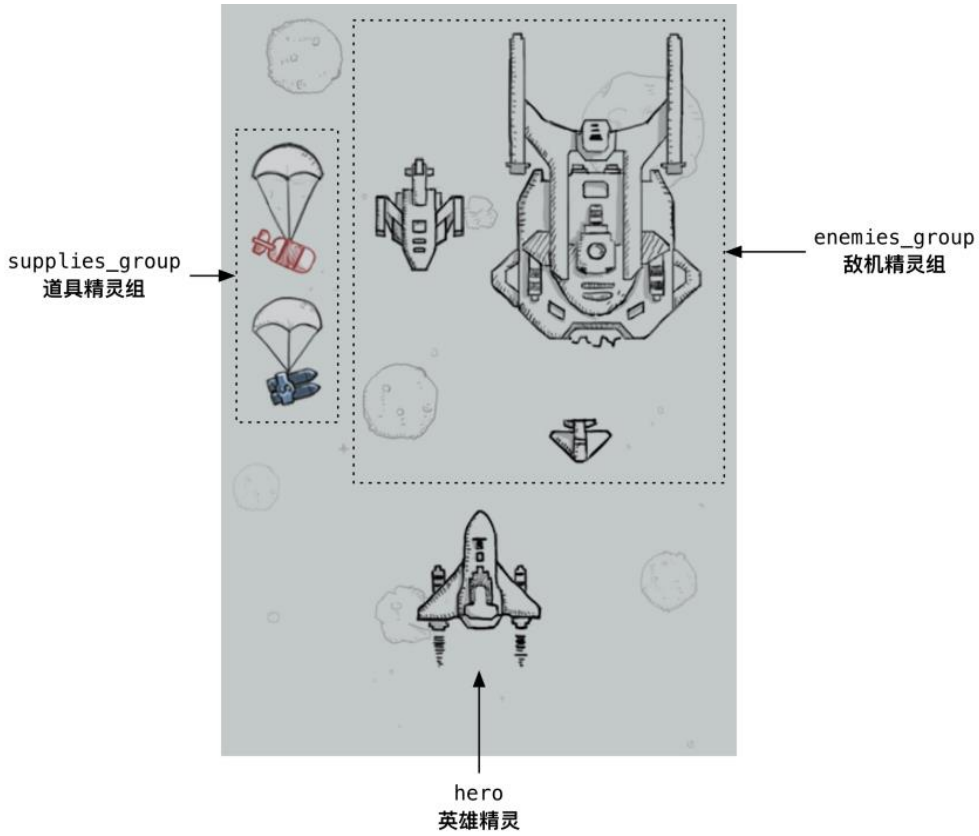
准备工作完成之后，便可以进入项目的实现阶段。游戏框架搭建是实现项目的第一步，它会按照游戏的完整流程搭建整个框架，如此便可以直接向框架内填充游戏的内容。



11.3.1 游戏类的设计

游戏类（Game）负责整个游戏的流程，它需要包含游戏中的主要元素，设计后的类图如图所示。

Game
main_window 游戏主窗口 is_game_over 结束标记 is_pause 暂停标记 hero 英雄精灵 hud_panel 指示器面板 player 音乐播放器 all_group 所有精灵组 enemies_group 敌机精灵组 supplies_group 道具精灵组
reset_game 重置游戏 create_enemies 创建敌机 create_supplies 创建道具 start 开始游戏 event_handler 事件监听 check_collide 碰撞检测



11.3.1 游戏类的设计

关于游戏类的属性和方法的说明具体如下。

1．Game类的属性

Game类的属性按作用的不同可以分为游戏属性和精灵（表示显示图像的对象，游戏窗口中看到的每个单独图像或者一行文本都可以看作一个精灵，例如，英雄飞机、一颗子弹、分数标签等）组属性。

11.3.1 游戏类的设计

（1）游戏属性

属性	说明
main_window	游戏主窗口，初始大小为 (480, 700)
is_game_over	游戏结束标记，初始为 False
is_pause	游戏暂停标记，初始为 False
hero	英雄精灵，初始显示在游戏窗口中间靠下位置
hud_panel	指示器面板，负责显示与游戏状态以及数据相关的内容，包括状态图像、游戏得分、炸弹数量、英雄命数，以及游戏暂停或结束时显示在游戏窗口中央位置的提示信息等
player	音乐播放器，负责播放背景音乐和游戏音效

11.3.1 游戏类的设计

（2）精灵组属性

游戏类中定义的精灵组属性如表所示。

属性	说明
all_group	所有精灵组，存放所有要显示的精灵，用于屏幕绘制和更新位置
enemies_group	敌机精灵组，存放所有敌机精灵对象，用于检测子弹击中敌机以及敌机撞击英雄
supplies_group	道具精灵组，存放所有道具精灵对象，用于检测英雄飞机拾取道具

11.3.1 游戏类的设计

关于游戏类的属性和方法的说明具体如下。

2.Game类的方法

Game类封装了多个方法，分别用于创建游戏元素、管理游戏的流程、监听系统事件等。

11.3.1 游戏类的设计

Game类中定义的方法如表所示。

方法	说明
reset_game()	重置游戏。在开启新一轮游戏之前，将游戏属性恢复到初始值
create_enemies()	创建敌机精灵。在新游戏开始或者关卡晋级后，根据当前游戏级别创建敌机精灵
create_supplies()	创建道具。游戏开始后每隔 30 秒随机投放炸弹补给或子弹增强道具
start()	开始游戏。创建时钟对象并且开启游戏循环，在游戏循环中监听事件、更新精灵位置、绘制精灵、更新显示、设置刷新帧率
event_handler()	事件监听。监听并处理每一次游戏循环执行时发生的事件，避免游戏循环中的代码过长
check_collide()	碰撞检测。监听并处理每一次游戏循环执行时是否发生精灵与精灵之间的碰撞，例如，子弹击中敌机、英雄拾取道具、敌机撞击英雄等

11.3.2 游戏框架实现

明确了游戏类的设计之后，便可以开始游戏框架的搭建工作。游戏框架的实现过程如下。

1.定义游戏窗口尺寸的全局变量

2.实现Game类的基础代码

3.在主程序中启动游戏

4．使用空格键切换游戏状态

目录页

11.4 游戏背景和英雄飞机

飞机大战项目需要管理众多游戏对象（例如敌机、英雄飞机、子弹等），并且实现众多动画（例如英雄飞机飞行动画，飞机碰撞动画等），使用精灵和精灵组来管理这些游戏对象的显示和动画效果是再合适不过了。

接下来，本节为大家介绍游戏精灵类，带领大家绘制游戏背景和英雄飞机，并实现游戏背景的滚动动画。

11.4.1 介绍精灵和精灵组

pygame专门提供了两个类：Sprite和Group，分别表示精灵和精灵组，其中精灵代表显示图像的游戏对象，精灵组代表保存和管理一组精灵的容器。Sprite和Group类中包含一些便于操作的属性或方法。

11.4.1 介绍精灵和精灵组

1.Sprite类

Sprite是一个表示游戏对象的基类，该类中提供的常用属性和方法如表所示。

类型	名称	说明
属性	image	记录从磁盘加载的图片内容或者字体渲染的文本内容，注意：子类中必须指定
	rect	记录 image 要显示在游戏窗口的矩形区域，注意：子类中必须指定
方法	update(*args)	默认什么都不做，子类可以根据需求重写，以改变精灵的位置 rect 或者显示内容image
	add(*groups)	将精灵添加到指定的精灵组（多值）
	remove(*groups)	将精灵从指定的精灵组（多值）移除
	kill()	将精灵从所有精灵组移除

值得一提的是，Sprite只是一个基类，实际开发中需要派生一个Sprite的子类，通过子类创建精灵对象。

11.4.1 介绍精灵和精灵组

2.Group类

Group是一个包含若干精灵的容器类，该类中提供了一些管理精灵的常用方法，关于这些方法的说明如表所示。

名称	说明
update(*args)	组内所有精灵调用update(*args) 方法，一次改变所有精灵的位置rect 或显示内容image
draw(surface)	将组内所有精灵的 image 绘在 surface 的 rect 矩形区域，实现在游戏窗口中一次绘制多个精灵的功能
sprites()	返回精灵组中所有精灵的列表
add(*sprites)	向精灵组中添加指定的精灵（多值）
remove(*sprites)	从精灵组中移除指定的精灵（多值）
empty()	清空精灵组中所有的精灵

11.4.2 派生游戏精灵子类

按照pygame官方文档的建议，我们在实际开发中不能直接使用Sprite类，而是要使用Sprite类派生的子类。

pygame官方文档规定了Sprite类子类的注意事项，具体如下。

- 子类可以重写update()方法。
- 子类必须为image和rect属性赋值。
- 子类的构造方法能够接收任意数量的精灵组对象，以将创建完的精灵对象添加到指定的精灵组中。
- 子类的构造方法中必须调用父类的构造方法，如此才能向精灵组中添加精灵。

11.4.2 派生游戏精灵子类

在game_items模块中定义pygame.sprite.Sprite的子类GameSprite，GameSprite类的类图如图所示。

GameSprite
image 图像 rect 矩形区域 speed 移动速度
__init__(self, image_name, speed, *groups) update(self, *args) 默认以 speed 为速度在垂直方向移动

GameSprite类定义了飞机大战游戏项目中大部分精灵类的通用特征和功能，它将作为其他精灵类的基类来使用。

11.4.3 绘制游戏背景和英雄飞机

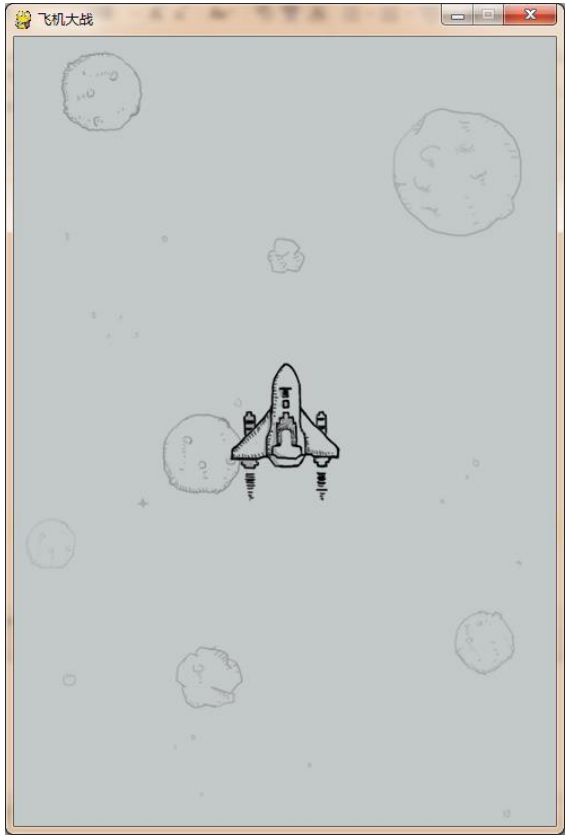
1.按照Game类的类图，我们需要在Game类的构造方法中：

- 创建 3 个精灵组
- 创建背景精灵
- 创建英雄精灵

2.在判断游戏状态之后、更新显示之前需要增加负责绘制和更新所有精灵的代码。

11.4.3 绘制游戏背景和英雄飞机

运行游戏，可以在游戏窗口中看到缓缓向下移动的星空背景和英雄飞机，如图所示。

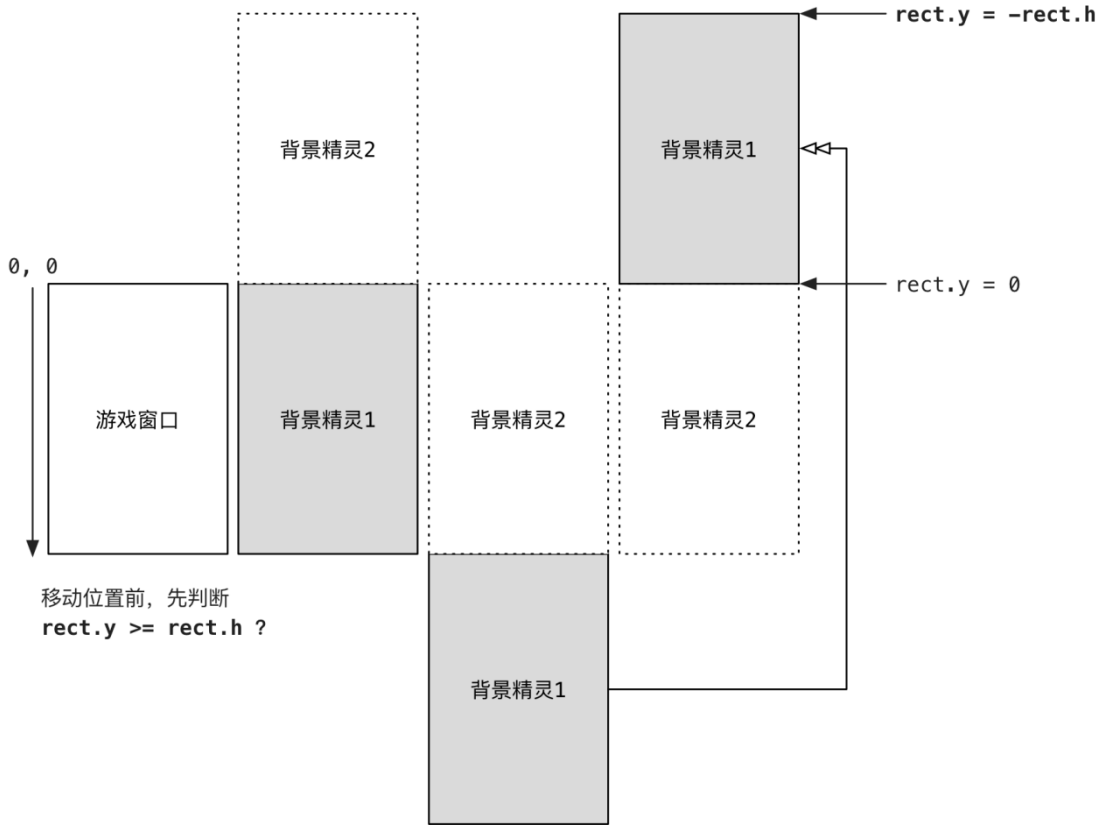


11.4.3 绘制游戏背景和英雄飞机

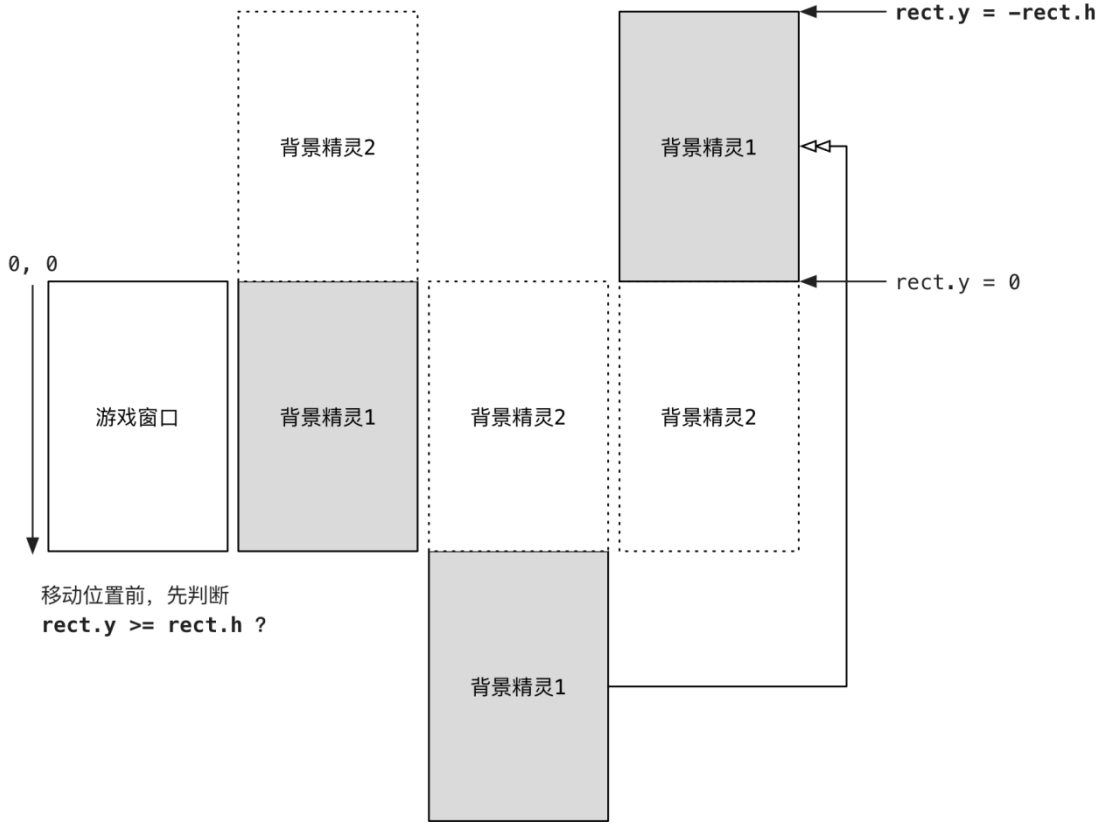
我们运行上个小节实现的程序可以发现，游戏刚刚启动时，它的背景图像是填满整个游戏窗口的，但是随着背景图像缓缓向下移动，游戏窗口上方出现了一个越来越大的灰色无图片区域。那么，如何才能实现背景图像持续滚动的效果呢?

11.4.3 绘制游戏背景和英雄飞机

答案是**使用两张背景精灵**：背景精灵1和背景精灵2，这两张背景精灵的实现原理如图所示。



11.4.3 绘制游戏背景和英雄飞机



这张图描述了背景图像持续滚动的实现原理，具体步骤如下。

（1）背景精灵1显示在屏幕上，与屏幕重合；背景精灵2放在屏幕的正上方，与背景精灵1连接起来。

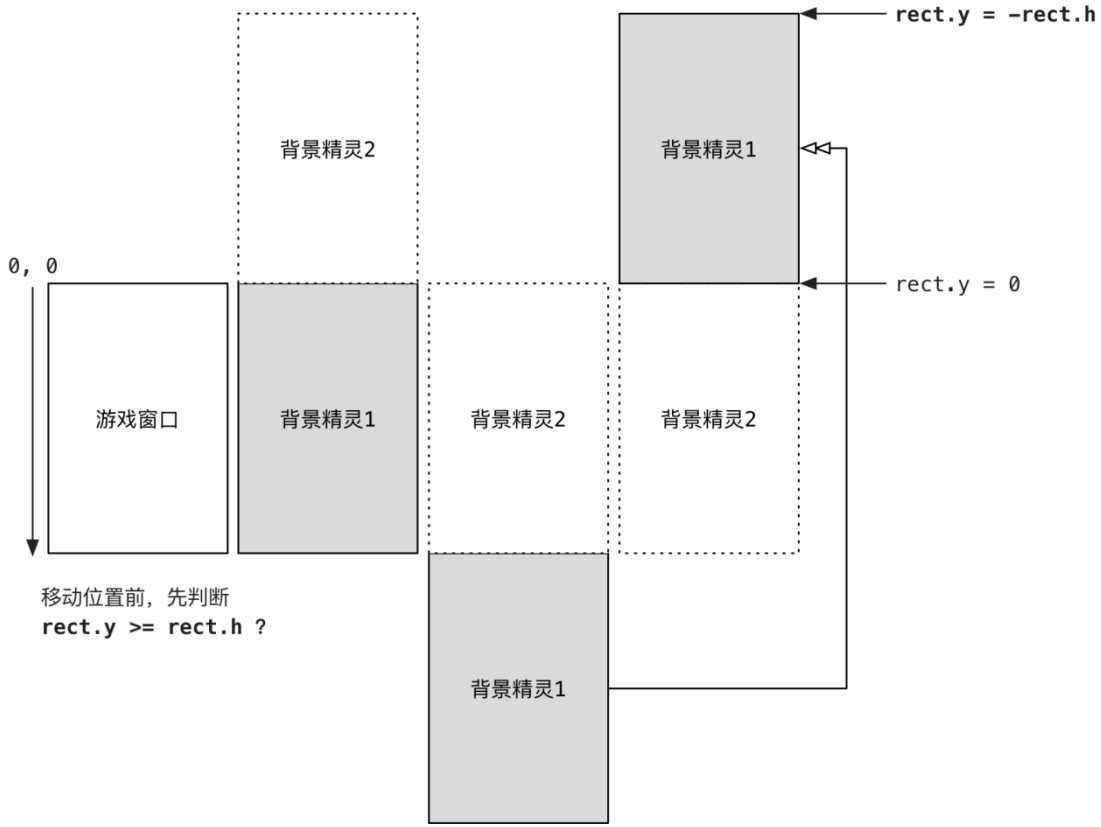
（2）两个背景精灵同时向下移动。

（3）当背景精灵1移出屏幕外时，背景精灵2正好全部显示在屏幕上。

（4）背景精灵1立即移动到屏幕的上方，与背景精灵2连接。

（5）两个背景精灵继续同时向下移动，回到步骤（2）。

11.4.3 绘制游戏背景和英雄飞机

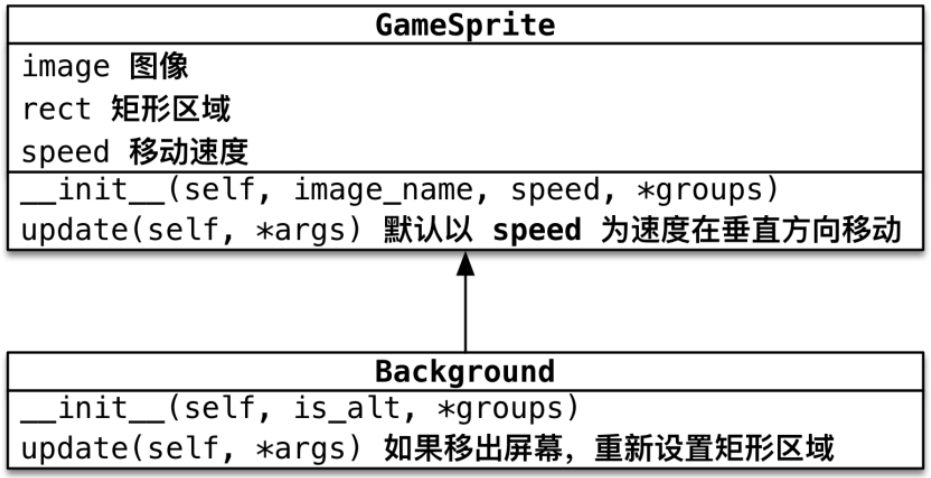


基于以上原理实现代码时：

- 可以通过判断背景图像的y值是否大于或等于屏幕的h值来判断背景图像是否移出屏幕下方，
- 通过将背景图像的y值设置为图像高度的负值将背景图像移动到屏幕上方。

11.4.3 绘制游戏背景和英雄飞机

理解了背景图像的实现原理之后，设计表示背景图像的背景精灵类Background。Background类的类图如图所示。



目录页

11.5 指示器面板

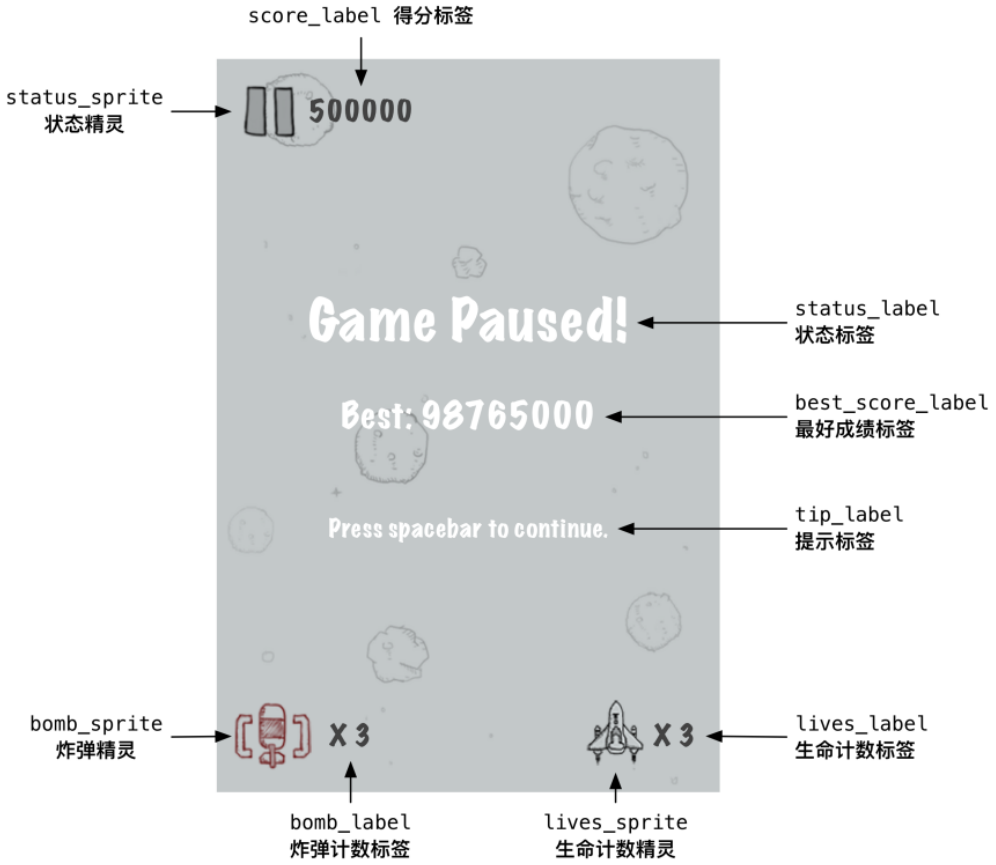
指示器面板又称HUD（平视显示器，全称为Head Up Display），它是航空器上的飞行辅助仪器。游戏开发中借鉴了指示器面板概念，把游戏相关的信息以类似指示器面板的方式显示在游戏画面上，可以让玩家随时了解那些最重要、最直接的游戏信息。

飞机大战负责显示与游戏状态和数据相关的内容，包括状态图像、游戏得分、炸弹图像、炸弹数量、英雄命数，以及在游戏暂停或结束时显示在游戏窗口中央位置的提示信息等。

11.5.1 指示器面板类的设计

指示器面板类HudPanel的类图如图所示。

HudPanel
score 游戏得分 lives_count 生命计数 level 游戏级别 best_score 最好成绩 status_sprite 状态精灵(暂停/继续) bomb_sprite 炸弹精灵 lives_sprite 生命计数精灵 score_label 得分标签 bomb_label 炸弹计数标签 lives_label 生命计数标签 status_label 状态标签 best_label 最好成绩标签 tip_label 提示标签
reset_panel 重置面板 show_bomb 显示炸弹数量 show_lives 显示生命计数 increase_score 增加得分 load_best_score 加载最好成绩 save_best_score 保存最好成绩 panel_paused 面板暂停 panel_resume 面板恢复



11.5.1 指示器面板类的设计

关于指示器面板类的属性和方法的说明如下。

1．指示器面板类的属性

指示器面板类的属性可以分为游戏数据属性和精灵属性：

- 游戏数据属性是一些指示器面板中变化的数据。
- 精灵属性是一些指示器面板中显示图像的精灵。

11.5.1 指示器面板类的设计

（1）游戏数据属性

指示器面板类中封装了4个与游戏数据相关的属性，关于这些属性的说明如表所示。

属性	说明
score	游戏得分，初始为 0
lives_count	英雄的生命计数，初始为 3
level	游戏级别，初始为 1
best_score	最好成绩，保存在 record.txt 文件中

11.5.1 指示器面板类的设计

（2）精灵属性

指示器面板类中封装了3个图像精灵和6个标签精灵属性，关于这些精灵属性的说明如表所示。

属性	说明
status_sprite	状态精灵，显示在游戏窗口左上角
bomb_sprite	炸弹精灵，显示在游戏窗口左下角
lives_sprite	生命计数精灵，显示在游戏窗口右下角
score_label	得分标签，显示在状态精灵的右侧
bomb_label	炸弹计数标签，显示在炸弹精灵的右侧
lives_label	生命计数标签，显示在生命计数精灵的右侧
best_label	最好成绩标签，显示在游戏窗口中间
status_label	状态标签，显示在最好成绩上方，文字为 Game Over! 或者 Game Paused!
tip_label	提示标签，显示在最好成绩下方

11.5.1 指示器面板类的设计

2．指示器面板类的方法

指示器面板类中封装了众多负责显示和更新数据、游戏状态的方法，关于这些方法的说明如表所示。

方法	说明
reset_panel()	重置面板数据。开启新一轮游戏之前，将游戏数据属性恢复为初始值
show_bomb()	显示炸弹数量。根据传入的炸弹数量，更改炸弹计数标签显示
show_lives()	显示生命计数。使用lives_count属性值更新生命计数标签显示
increase_score()	增加分数。根据消灭的敌机分值增加游戏得分、计算是否奖励生命、调整最好成绩以及计算游戏级别，并且更改分数标签显示
reset_panel()	重置面板数据。开启新一轮游戏之前，将游戏数据属性恢复为初始值
load_best_score()	加载最好成绩。从record.txt文件中加载最好成绩
save_best_socre()	保存最好成绩。将最好成绩保存到 record.txt文件中
panel_paused()	面板暂停。当游戏暂停/结束时，显示游戏窗口中间位置的游戏暂停/结束的提示信息
panel_resume()	面板恢复。当游戏运行时，隐藏游戏窗口中间位置的提示信息

11.5.2 指示器面板类的准备

明确了指示器面板类的设计之后，接下来我们要完成指示器面板类的准备工作，包括：

1. 创建指示器面板类。
2. 设计状态按钮类。
3. 创建图像精灵。

11.5.2 指示器面板类的准备

1．创建指示器面板类

在game_hud模块中定义HudPanel类，并实现构造方法。

2．设计状态按钮类

按照游戏说明，玩家按下空格键暂停或继续游戏时，游戏窗口左上角状态精灵的显示图像会发生相应的变化。为了方便后续的代码，我们可以从GameSprite类派生一个子类StatusButton类，通过StatusButton类处理状态图像的切换。

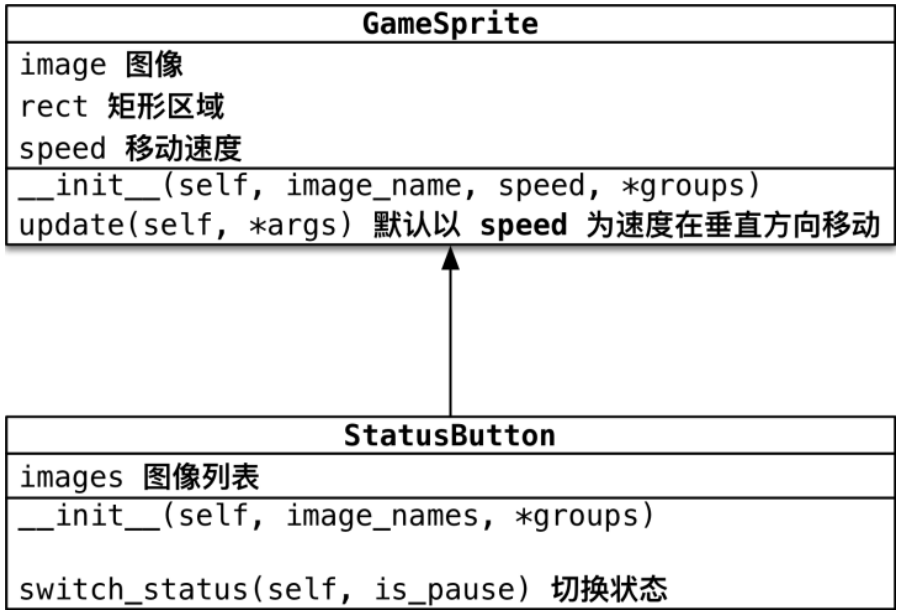
11.5.2 指示器面板类的准备

1．创建指示器面板类

在game_hud模块中定义HudPanel类，并实现构造方法。

2．设计状态按钮类

StatusButton类的类图如图所示。



11.5.2 指示器面板类的准备

3．创建图像精灵

在HUDPanel类的构造方法中创建指示器面板中的图像精灵，包括左上角的状态精灵（暂停 / 继续）、左下角的炸弹精灵、以及右下角的生命计数（小飞机）精灵。

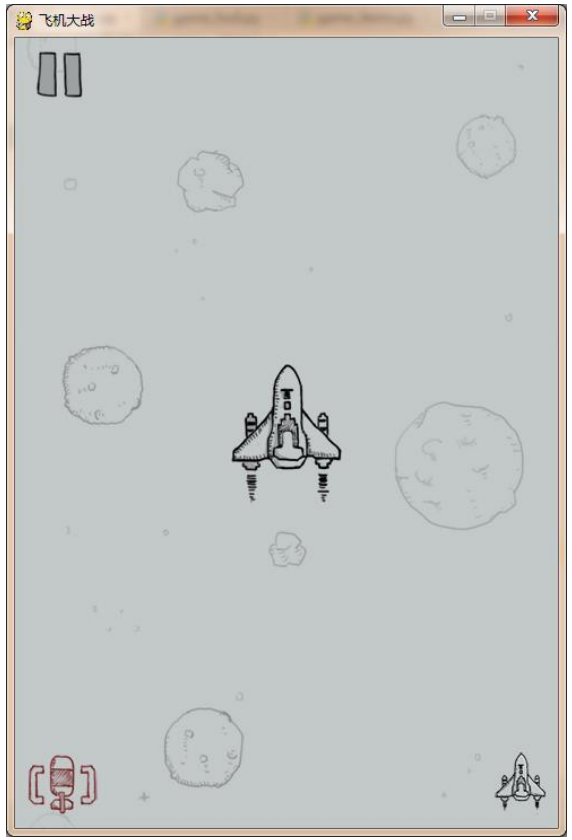
① 首先，在构造方法的上方定义几个类属性，以方便后续可以设置精灵的矩形区域和标签精灵的字体颜色。

② 然后，在构造方法的末尾创建精灵并设置显示位置。

③ 最后，在Game类的构造方法末尾创建指示器面板。

11.5.2 指示器面板类的准备

运行游戏，可以看到游戏窗口的左上方、左下方以及右下方位置分别显示了暂停图像、炸弹图像和小飞机的生命指示图像。如图所示。



11.5.3 使用精灵实现文本标签

上个小节运行的游戏窗口已经显示了暂停图像、炸弹图像和小飞机的生命指示图像。接下来我们实现文本标签的相关功能，包括：

1. 使用自定义字体定义文本标签精灵。
2. 创建指示器面板的文本标签精灵。

11.5.3 使用精灵实现文本标签

1．使用自定义字体定义文本标签精灵

pygame的font模块提供了SysFont类，使用SysFont类可以创建系统字体对象，实现在游戏窗口中绘制文字内容。但系统字体存在以下几点限制：

- 不够美观。一般操作系统默认提供的字体大多比较规整，直接应用到游戏中，呈现的效果比较呆板。
- 不支持跨平台。不同操作系统使用的系统字体的名称不同，当一个程序被移植到其他类型的操作系统运行时，可能会导致文字内容无法被正确显示。

11.5.3 使用精灵实现文本标签

1．使用自定义字体定义文本标签精灵

在游戏开发中既要拥有美观的字体，又要支持跨平台，使用自定义字体会是一个不错的选择。

使用自定义字体就是把字体文件和程序文件保存在同一个目录下。当程序执行时，会直接加载并使用目录下的字体文件；当程序移植时，会复制字体文件。

11.5.3 使用精灵实现文本标签

1．使用自定义字体定义文本标签精灵

pygame的font模块还提供了另外一个Font类，使用Font类可以创建自定义字体对象。

我们在game_items模块中派生一个pygame.sprite.Sprite类的子类Label。

Label类表示文本标签，其类图如图所示。

Label
font 字体对象 color 文本的颜色 image 文本内容渲染生成的图像 rect 文本图像矩形区域
__init__(self, text, size, color, *groups) set_text(self, text) 使用 text 重新渲染并更新 rect

11.5.3 使用精灵实现文本标签

1．使用自定义字体定义文本标签精灵

需要注意的是，因为文本标签的内容并不需要在每一次游戏循环执行时发生变化，所以Label类无需重写update()方法。在游戏执行过程中，如果希望更改文本标签的显示内容，可以通过set_text()方法重新渲染文本图像，并且更新矩形区域。

11.5.3 使用精灵实现文本标签

2．创建指示器面板的文本标签精灵

在HUDPanel类的构造方法末尾创建指示器面板的文本标签精灵。

值得一提的是，在设定标签精灵的位置时：

- 既可以参照之前的游戏屏幕示意图，依次根据选好的每一个标签精灵的参照物设定；
- 也可以每设置一次标签精灵的位置后进行运行测试，以确保每个标签精灵具有正确的位置。

11.5.3 使用精灵实现文本标签

运行游戏，可以在游戏主窗口的左上、左下以及中间位置看到图像对应的文本内容，如图所示。

11.5.3 使用精灵实现文本标签

此时的指示器面板存在一些问题：

- 游戏得分、炸弹计数、生命计数的数据不会变化；
- 最好成绩始终为0；
- 提示信息不会随着游戏状态的变化而变化，而是始终显示在游戏窗口中央位置。

关于指示器面板的问题，我们会在后续的小节中逐一解决。

11.5.4 显示和修改游戏数据

HUDPanel类封装3个方法：show_bomb()、show_lives()、increase_score()，分别实现显示炸弹数量、显示生命计数、增加游戏得分的功能。

1．显示炸弹数量

按照游戏说明，英雄飞机出场后默认会携带3颗炸弹，且会在玩家按下字母b时引爆炸弹。引爆炸弹后，游戏窗口左下角的炸弹数量会减少。

- 首先，在HUDPanel类中实现show_bomb()方法，通过show_bomb()方法的参数更新炸弹计数标签的显示。
- 然后，在game模块的顶部导入random模块。
- 最后，在Game类的event_handler()方法中添加代码，监听玩家按下字母b事件，并使用随机数测试显示的炸弹数量。

11.5.4 显示和修改游戏数据

1．显示炸弹数量

运行游戏，按下字母b，可以在游戏窗口的左下角看到炸弹数量的变化，如图所示。

11.5.4 显示和修改游戏数据

2．显示生命计数

按照游戏说明，英雄飞机被敌机撞毁后，游戏窗口右下角的生命计数应该相应减少；英雄飞机每得到100000分会被奖励1条命，游戏窗口右下角的生命计数应该相应增加。

- 在HUDPanel类中定义show_lives()方法，使用生命计数属性值更新生命计数标签的显示即可。
- 修改Game类的event_handler()方法，在监听到玩家按下字母b的事件处理中，使用随机数测试生命计数的显示。

运行游戏，按下字母b，可以在游戏窗口的右下角看到生命计数的变化。

11.5.4 显示和修改游戏数据

3．增加游戏得分

按照游戏说明，每当英雄飞机摧毁一架敌机之后，游戏得分应该相应增加，且根据摧毁的敌机类型增加相应的分值。需要注意的是，随着游戏分数的增加，其他属性可能会受到影响，包括：

（1）生命计数。英雄飞机每得到100000分会被奖励1条命。

（2）最好成绩。若当前游戏得分超过了历史最好成绩，将当前的游戏得分设置为最好成绩。

（3）游戏级别。级别不同，游戏中敌机的类型、数量和速度不同。

3．增加游戏得分

因此，在增加游戏得分时，除了要完成根据摧毁敌机的分值增加游戏得分属性的功能之外，还需处理与之相关的生命计数、最好成绩以及游戏级别。

- 首先，在HUDPanel类的构造方法上方定义几个类属性，以方便后续计算奖励生命和游戏级别。
- 然后，在HUDPanel类中实现increase_score()方法。
- 最后，在Game类的游戏循环中增加测试increase_score方法的代码。

11.5.4 显示和修改游戏数据

3．增加游戏得分

运行游戏，可以看到游戏窗口左上角的分数标签快速地发生变化，达到奖励分数后，游戏窗口右下角的生命计数数值增加，如图所示。

11.5.4 显示和修改游戏数据

3．增加游戏得分

运行游戏，可以看到游戏窗口左上角的分数标签快速地发生变化，达到奖励分数后，游戏窗口右下角的生命计数数值增加，如图所示。

此外，游戏级别提升后，控制台输出的信息如下所示：

```
升级到 2
升级到 3
```

11.5.5 保存和显示最好成绩

最好成绩是一个永久成绩，它不会随游戏的退出而消失，因此需要持久化存储。

因为存储的数据量较小，所以直接使用文件存储即可。

在HUDPanel类的类图中，设计了save_best_score()方法和load_best_score()方法，分别用于保存和加载最好成绩。下面分别实现save_best_score()方法和load_best_score()方法，并在适当的时机调用这两个方法，实现最好成绩的保存和显示。

11.5.5 保存和显示最好成绩

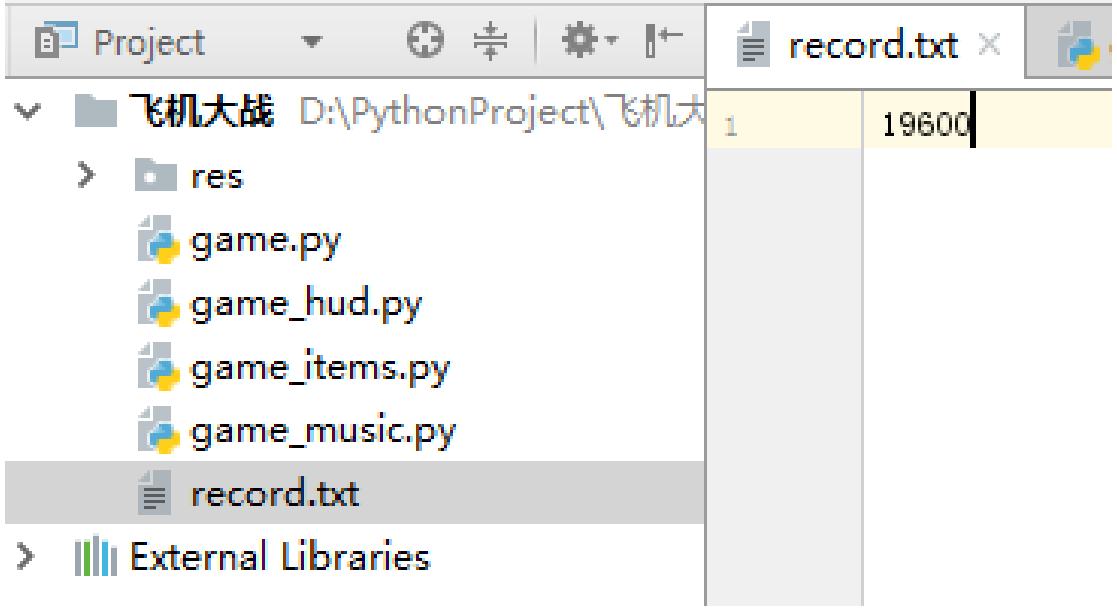
1．保存最好成绩

- 首先，在HUDPanel类的构造方法上方定义一个类属性，使用该属性指定保存最好成绩的文件名。
- 然后，在HUDPanel类中实现save_best_score()方法。
- 最后，修改Game类的游戏循环，在游戏退出之前，调用指示器面板的save_best_score()方法。

11.5.5 保存和显示最好成绩

1．保存最好成绩

运行游戏，游戏分数增长到一定程度后，按下ESC键退出游戏，可以看到飞机大战项目目录下新建了record.txt文件，record.txt文件中保存了退出游戏时的分数。如图所示。



11.5.5 保存和显示最好成绩

2．加载最好成绩

- 在HUDPanel 类中实现load_best_score()方法。
- 修改HUDPanel类的构造方法，在定义完游戏属性之后，调用load_best_score()方法。

11.5.5 保存和显示最好成绩

2．加载最好成绩

运行游戏，可以看到游戏窗口的中间位置显示了之前保存的最好成绩，如图所示。

11.5.6 保存和显示最好成绩

游戏的状态一共有三种：进行、暂停和结束。在不同的游戏状态下，指示器面板上显示不同的提示信息。

指示器面板按照游戏的状态可以分为暂停面板和恢复面板，其中：

- 暂停面板是游戏处于暂停或者结束状态的面板。
- 恢复面板是游戏处于进行状态的面板。

当游戏暂停或者结束时，游戏窗口的中央位置会显示提示信息，其他位置的数据不会发生变化；当游戏进行时，游戏窗口的中央位置不会显示提示信息，其他位置的数据会随着游戏的推进而变化。

下面先带领大家理清精灵组的绘制顺序，之后实现暂停和恢复面板。

11.5.6 保存和显示最好成绩

1．理解精灵组的绘制顺序

指示器面板上的文字应该显示在其他元素之上，也就是说，游戏窗口上显示的元素是有顺序的，那么应该怎么实现特定的显示效果呢？我们先观察Game类的构造方法中创建精灵的代码，具体如下：

```
# 创建精灵
# 背景精灵，交替滚动
self.all_group.add(Background(False), Background(True))
# 英雄精灵，静止不动
hero = GameSprite("me1.png", 0, self.all_group)
hero.rect.center = SCREEN_RECT.center  # 显示在屏幕中央
# 指示器面板
self.hud_panel = HUDPanel(self.all_group)
```

由以上代码可知，加入精灵组的顺序为“背景→英雄精灵→指示器面板”。

11.5.6 保存和显示最好成绩

2．实现暂停和恢复面板

无论暂停还是恢复面板，指示器面板的提示信息必须显示在界面的其他元素之上。为了保证实现这种显示效果，我们可以在创建指示器面板时只创建3个提示标签精灵的属性，暂时先不将这3个精灵添加到显示精灵组。

当游戏暂停或结束时，设置3个标签精灵的文字和显示位置，然后将标签精灵添加到显示精灵组；当游戏继续时，将3个标签精灵从显示精灵组中移除即可。这样做就可以保证这3个文本信息显示在整个界面的最上层。

11.5.6 保存和显示最好成绩

2．实现暂停和恢复面板

明确了思路之后：

- 首先修改HUDPanel类中构造方法的代码，在创建3个提示标签时不传递显示精灵组。
- 其次，在HUDPanel类中实现包含暂停面板操作的panel_pause()方法。
- 然后，在HUDPanel类中再实现包含恢复面板操作的panel_resume()方法。
- 最后，修改Game 类的游戏循环。在游戏结束或游戏暂停时，调用指示器面板的panel_pause()方法；在游戏进行时，调用指示器面板的panel_resume()方法。

11.5.6 保存和显示最好成绩

2．实现暂停和恢复面板

运行游戏，按下空格键后可以看到游戏窗口的中间位置显示了暂停状态的提示文字，继续按下空格键后隐藏了提示文字，如图所示。

11.5.7 游戏结束后重置面板

- 当英雄飞机因没有剩余命数而无法继续战斗时，游戏结束。
- 如果玩家按下空格键，那么会重新开启一轮新的游戏。

在新一轮游戏开始之前，我们应该重置指示器面板中与游戏数据相关的属性，并且更新对应的标签显示，否则会影响到新一轮游戏的数据处理。

11.5.7 游戏结束后重置面板

1．判断游戏结束

在Game类的游戏循环的开始位置增加一行代码，判断生命计数是否为0，若为0说明游戏结束。

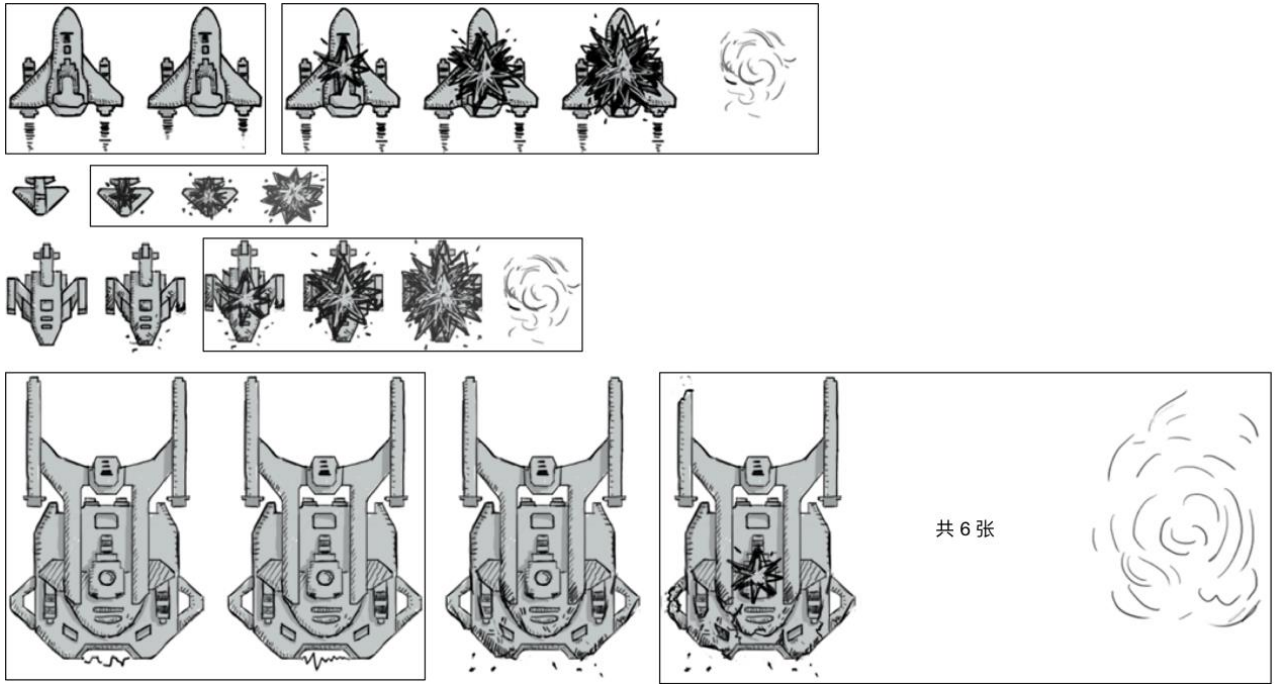
2．重置面板

在新一轮游戏开始之前，我们应该重新设置指示器面板中与游戏数据相关的属性，并且更新这些属性对应的标签。

目录页

11.6 逐帧动画和飞机类

逐帧动画（Frame By Frame）是一种常见的动画形式，它的原理是在每一次游戏循环执行时逐帧绘制不同的内容，形成连续播放的动画效果。GIF动图就是一种逐帧动画。



11.6 逐帧动画和飞机类

在飞机大战游戏中，飞机需要根据不同的状态设置对应的图片和动画。

飞机的状态一般包括3种，分别为：正常飞行状态、被击中受损状态和被摧毁状态。

其中英雄飞机和小敌机只有正常飞行状态和被摧毁状态，没有被击中受损状态，

这是因为英雄飞机一旦被撞就会牺牲，而小敌机一旦被击中就会毙命。

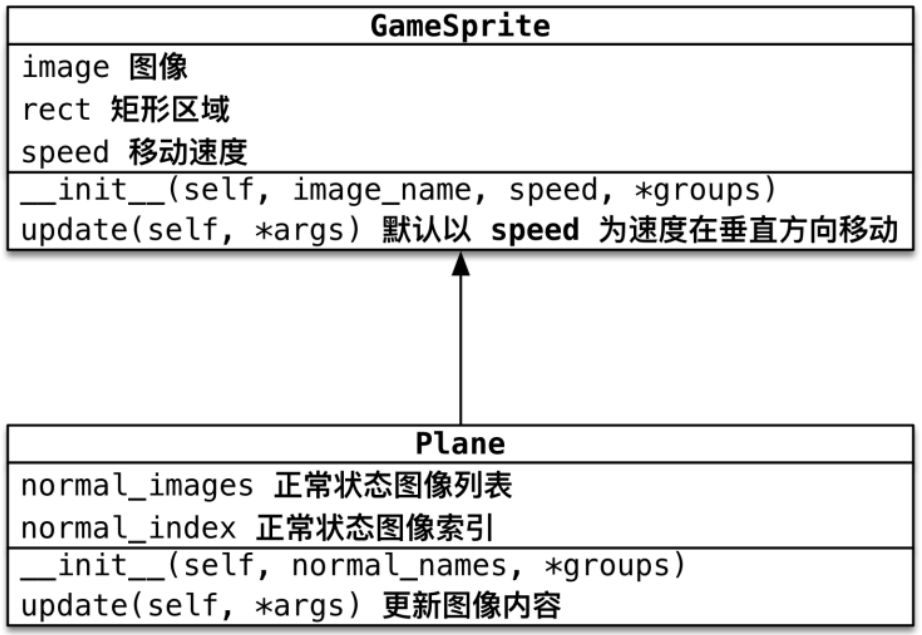
11.6.1 逐帧动画的基本实现

尽管逐帧动画在游戏开发的应用非常广泛，但是pygame并没有提供直接的实现方式。下面我们先学习一下如何使用pygame实现逐帧动画。

11.6.1 逐帧动画的基本实现

1．派生简单的飞机类

根据飞机的特征和行为设计飞机类Plane，并让Plane类继承GameSprite类。Plane类的类图如图所示。



11.6.1 逐帧动画的基本实现

2．设置逐帧动画频率

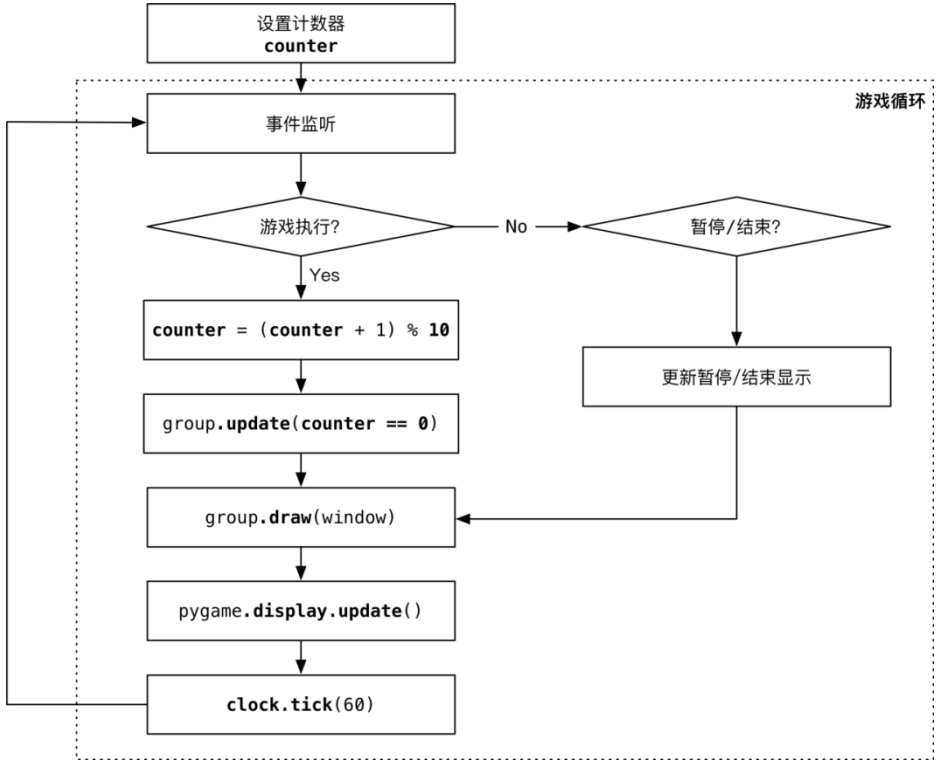
当游戏循环的刷新帧率设置为 60 帧/秒时，玩家在游戏中的交互才能够得到快速地响应。如果将刷新帧率设置得太低，玩家在交互时会感到明显的卡顿。但是，60帧/秒的刷新帧率对动画而言是非常快的。

如何才能做到既保证流畅的用户交互，又降低逐帧动画的动画帧率呢？要解决这个问题，这里可以引入一个计数变量，通过这个变量实现循环每执行10次才更换一张图像、每秒更换6次图像的功能，进而达到降低逐帧动画的频率的目的。

11.6.1 逐帧动画的基本实现

2．设置逐帧动画频率

降低逐帧动画的流程如图所示。



11.6.2 飞机类的设计与实现

本小节将分设计飞机类和改进飞机类两部分扩展之前完成的飞机类，以便基于Plane类创建游戏中的4种类型的飞机对象。

1. 设计飞机类

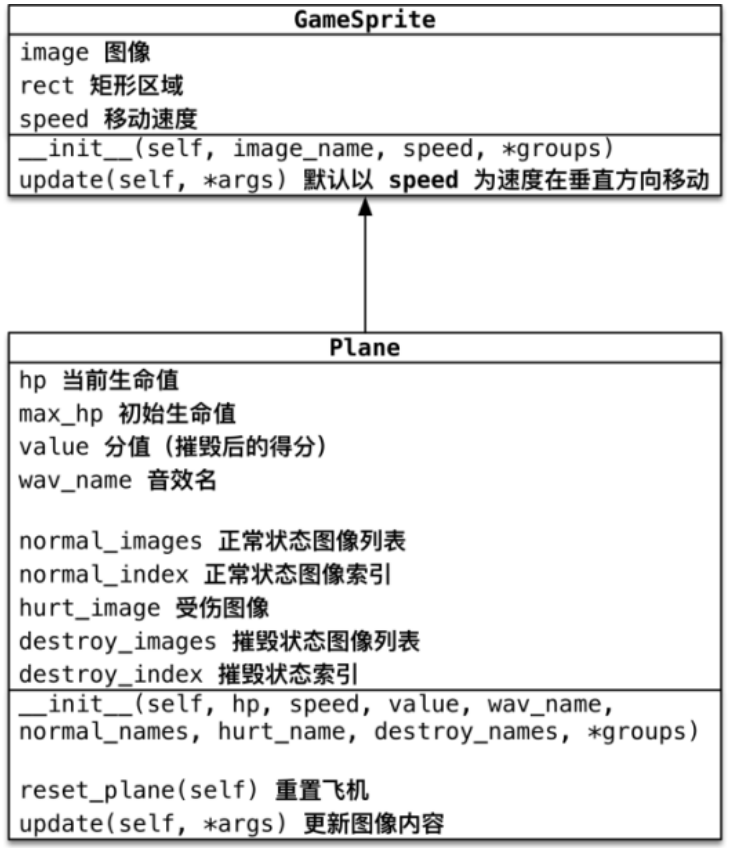
按照游戏介绍，飞机大战游戏包含4种类型的飞机，每种类型的飞机具有生命值、速度、分值、飞行动画、被击中图片、被撞毁动画、撞毁音效这些特征，关于4种飞机的特征如表所示。

名称	生命值	速度	分值	飞行动画	被击中图片	被摧毁动画	摧毁音效
英雄飞机	无	4	0	有	无	有	有
小敌机	1	1~7	1000	图片	无	有	有
中敌机	6	1~3	6000	图片	有	有	有
大敌机	15	1	15000	有	有	有	有

11.6.2 飞机类的设计与实现

1. 设计飞机类

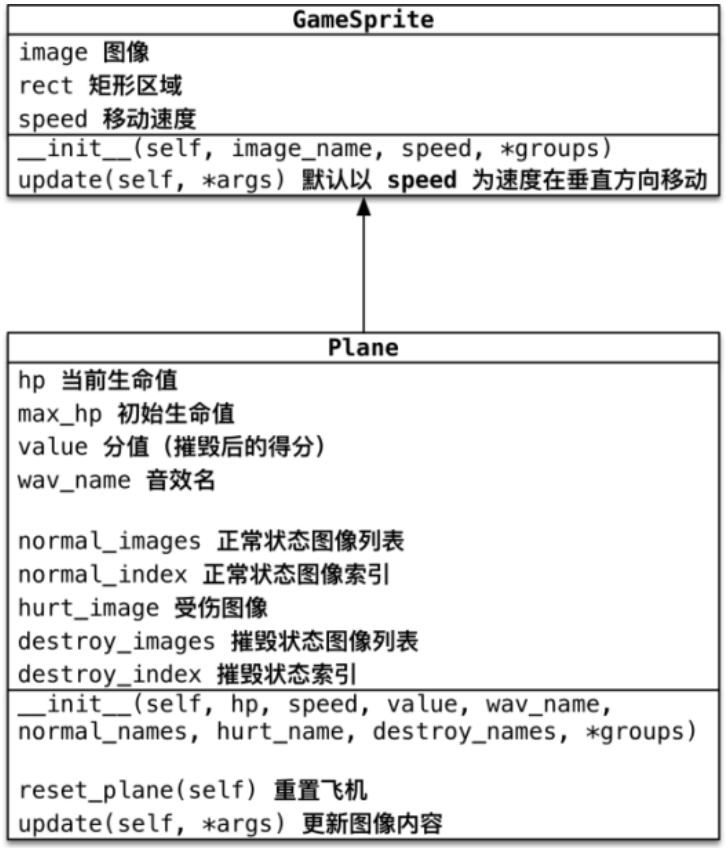
下面根据飞机的特征和行为飞机类（Plane）。Plane类的类图如图所示。



11.6.2 飞机类的设计与实现

1.设计飞机类

下面根据飞机的特征和行为飞机类（Plane）。Plane类的类图如图所示。



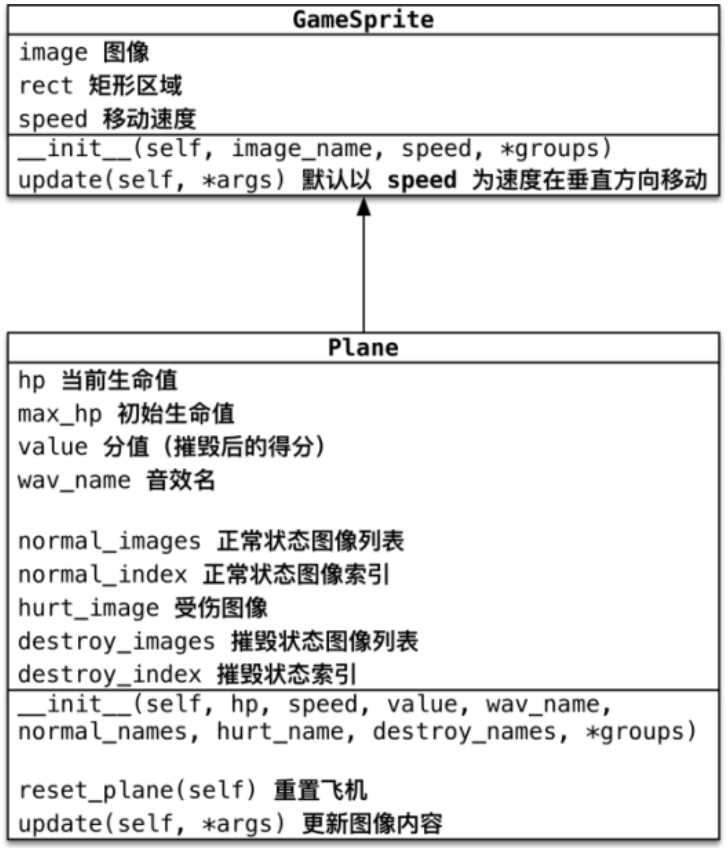
Plane类的属性包括hp、max_hp、value等，各个属性的说明如表所示。

属性	说明
hp	当前生命值。初始化时指定，被击中时会变化
max_hp	初始生命值。初始化时等于当前生命值，用于判断敌机是否受损，0 < hp < max_hp
value	分值。敌机被摧毁后的得分
wav_name	被摧毁时播放的音效文件名
normal_images	正常状态图像列表
normal_index	正常状态图像索引
hurt_image	受损图像（注意，所有类型的飞机在受损时都没有逐帧动画）
destroy_images	摧毁状态图像列表
destroy_index	摧毁状态图像索引

11.6.2 飞机类的设计与实现

1.设计飞机类

下面根据飞机的特征和行为飞机类（Plane）。Plane类的类图如图所示。



Plane类的方法包括reset_plane()和update()，关于这两个方法的说明如表所示。

方法	说明
reset_plane()	重置飞机，飞机被摧毁后，重新设置飞机的当前生命值及图像索引
update()	更新飞机，覆盖父类方法，根据参数设置飞机的显示图像，Plane 类中不考虑飞机位置的变化

11.6.2 飞机类的设计与实现

2.改进飞机类

① 按照Plane类的设计来改进飞机类的代码。

② 使用改进后的Plane类创建英雄飞机，并测试状态变化时显示的逐帧动画。

11.6.3 派生敌机子类

敌机与英雄飞机属于不同的阵营，它们在游戏介绍的设定上具有很大的差异，比如敌机初始出现在游戏窗口上方的随机位置，各自以不同的速度飞入游戏窗口，飞出游戏窗口后会重新设置初始位置；而英雄飞机在游戏窗口上的位置由玩家控制。

因此，需要以Plane类为基础，分别派生表示敌机和英雄飞机的不同子类。本小节先从Plane类派生一个表示敌机的子类。

1.设计敌机类

2.实现敌机类的基本功能

3.创建敌机

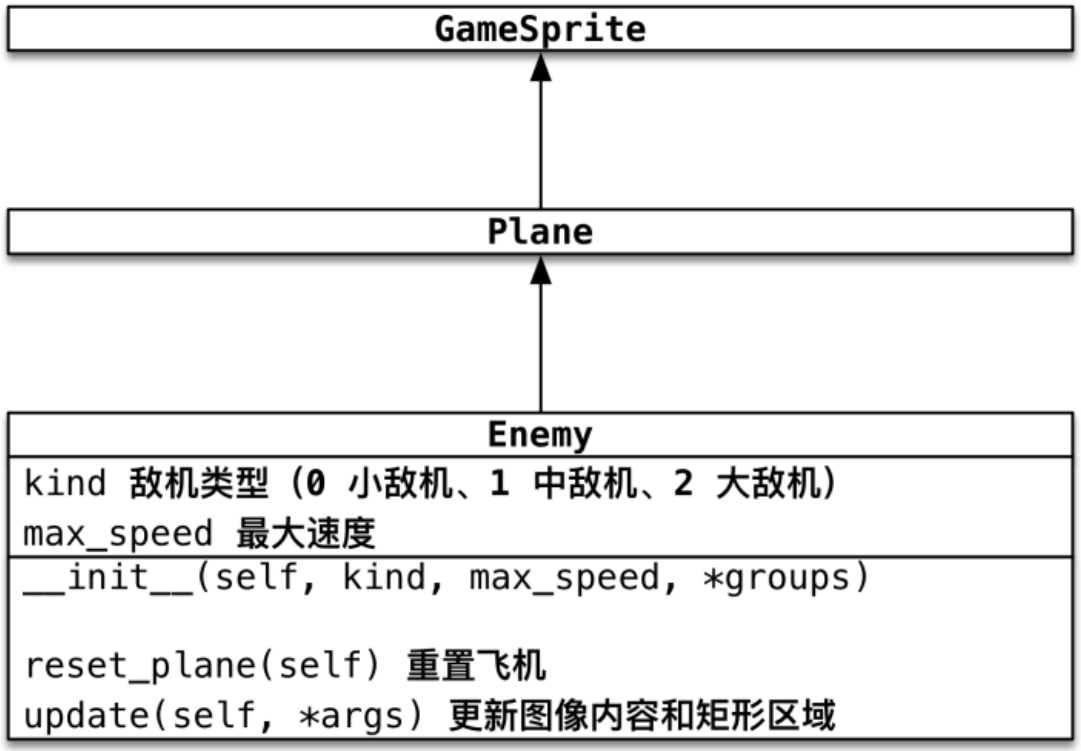
4．设置敌机精灵的随机位置

11.6.3 派生敌机子类

1．设计敌机类

敌机起初出现在游戏窗口上方的随机位置，按各自的速度沿垂直方向向下飞行，进入游戏窗口。若敌机飞出了游戏窗口，它会被设置为初始状态，重新出现在游戏窗口上方的随机位置。

敌机类Enemy的类图如图所示。



11.6.3 派生敌机子类

Enemy类中包括两个属性：kind和max_speed，关于这两个属性的说明如表所示。

属性	说明
kind	敌机类型。0代表小敌机；1代表中敌机；2代表大敌机
max_speed	最大速度

11.6.3 派生敌机子类

1．设计敌机类

Enemy类需要重写父类的reset_plane()方法和update()方法，关于这两个方法的说明如表所示。

方法	说明
reset_plane()	重置飞机。敌机被摧毁或者飞出游戏窗口后，重新设置飞机的位置以及速度
update()	更新飞机。调用父类方法设置飞机的显示图像，然后根据速度修改飞机的矩形区域，最后判断是否飞出游戏窗口，飞出游戏窗口则重置飞机

11.6.3 派生敌机子类

2．实现敌机类的基本功能

① 在game_items模块的顶部导入random模块，以方便使用随机数。

② 在game_items模块中定义继承Plane类的Enemy类。

11.6.3 派生敌机子类

3．创建敌机

敌机通过Game类中的create_enemies()方法创建，create_enemies()方法会根据不同的游戏级别来创建不同数量的敌机。关卡和敌机的数量关系如表所示。

名称	分值范围	小敌机数量（速度）	中敌机数量（速度）	大敌机数量（速度）
关卡 1	< 10000	16（1 ~ 3）	0（1）	0（1）
关卡 2	< 50000	24（1 ~ 5）	2（1）	0（1）
关卡 3	>= 50000	32（1 ~ 7）	4（1 ~ 3）	2（1）

11.6.3 派生敌机子类

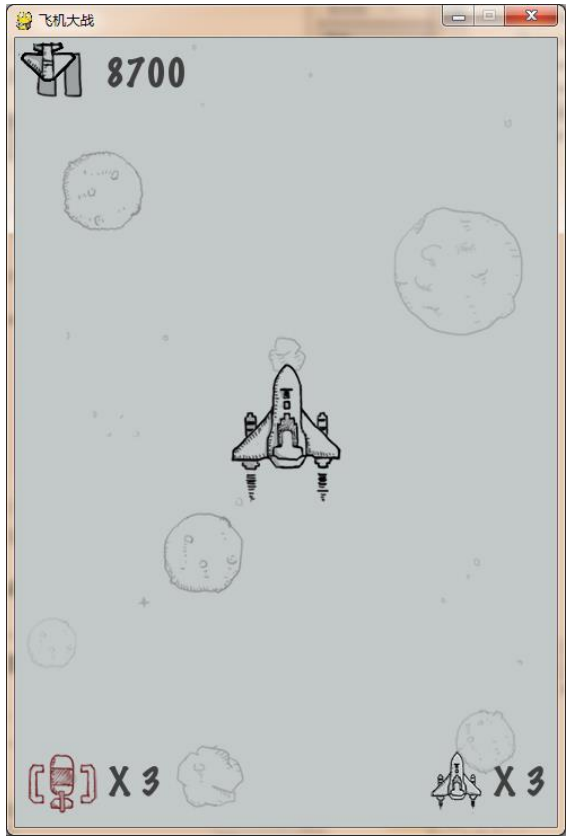
3．创建敌机

① 实现create_enemies()方法。

② 修改Game类的构造方法，在创建完指示器面板之后，调用create_enemies()方法创建敌机。

11.6.3 派生敌机子类

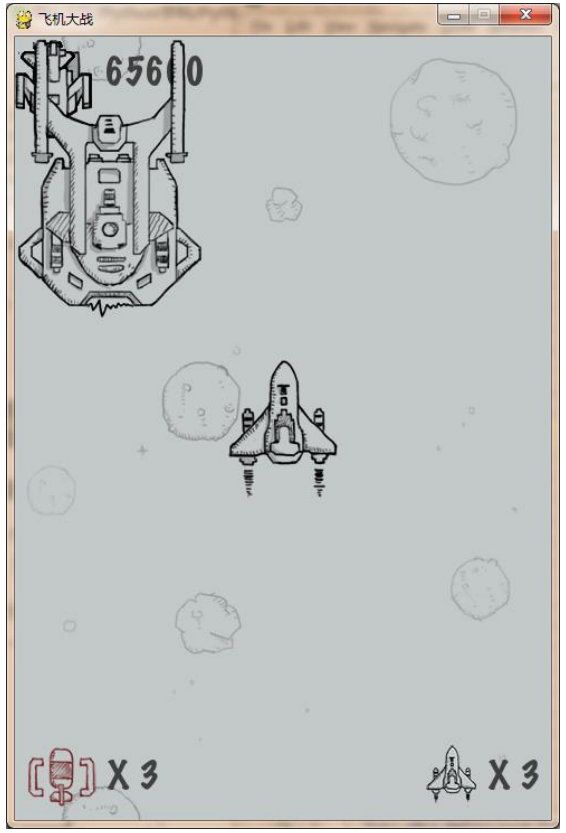
运行游戏，可以在游戏窗口的左上角看到1架敌机，如图所示。



图中之所以显示了1架敌机，是因为还没有设置敌机的随机位置，也就是说，第1关的16架小敌机都被绘制在游戏窗口左上角的同一个位置。

11.6.3 派生敌机子类

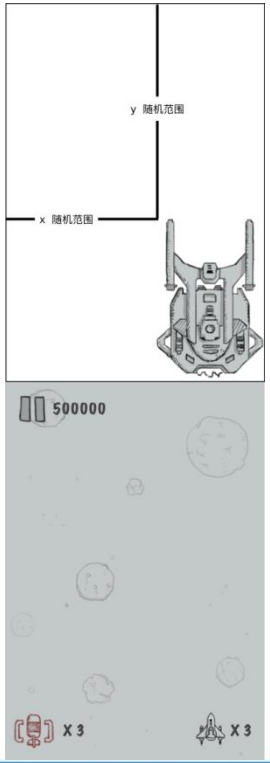
修改之前在游戏循环中增加的测试代码，在升级之后调用create_enemies()方法，以测试游戏级别提升后能否创建中敌机和大敌机。



11.6.3 派生敌机子类

4．设置敌机精灵的随机位置

按照游戏说明，敌机出现的位置是随机的，并且由游戏窗口的上方逐渐进入窗口。敌机出现位置的示意图如图所示。



由图可知，敌机的初始随机位置的设置思路如下。

（1）敌机出现的水平方向坐标取值在0 ~ (SCREEN_RECT.w - self.rect.w)之间，可以取该范围内的一个随机数，做为敌机矩形区域的x值。

（2）敌机出现的垂直方向坐标取值在0 ~ (SCREEN_RECT.h - self.rect.h)之间，可以取该范围内的一个随机数，再减去游戏窗口高度，做为敌机矩形区域的y值。

4．设置敌机精灵的随机位置

按照刚刚明确的思路，对Enemy 类的 reset_plane()方法进行调整。

为了方便测试敌机的初始位置以及后续的飞机爆炸，我们临时将设置y值的代码进行更改，改为只使用随机值，不减去屏幕高度，这样创建后的飞机都可以显示在游戏窗口中。

11.6.3 派生敌机子类

4．设置敌机精灵的随机位置

按照刚刚明确的思路，对Enemy 类的 reset_plane()方法进行调整。

运行游戏，可以看到游戏窗口出现了密密麻麻的小敌机，并且随着游戏级别的提升，增加了更多数量和种类的敌机。

11.6.3 派生敌机子类

4．设置敌机精灵的随机位置

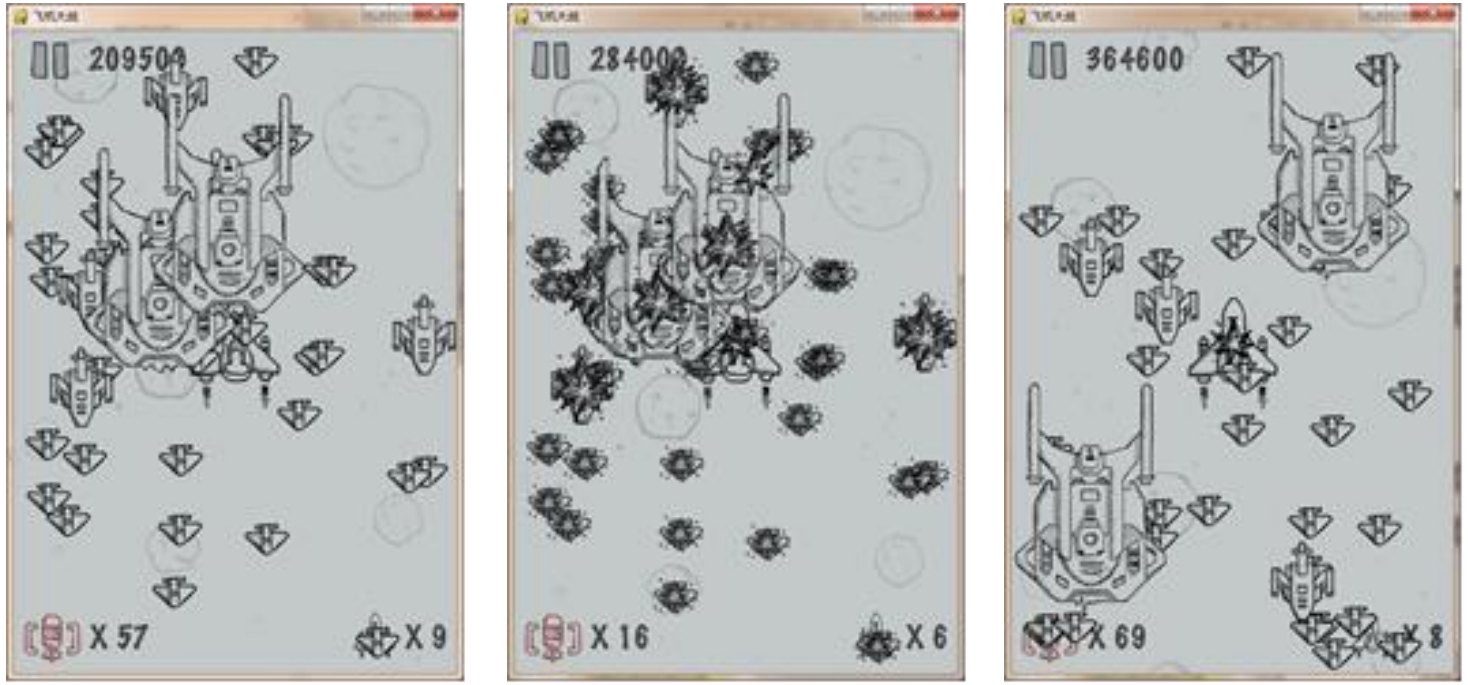
按照刚刚明确的思路，对Enemy 类的 reset_plane()方法进行调整。

修改之前在引爆炸弹的事件监听中增加的测试代码，模拟炸毁敌机的场景，测试当敌机被摧毁后初始位置是否会被重新设置。

11.6.3 派生敌机子类

4．设置敌机精灵的随机位置

运行游戏，按下字母b，可以看到游戏窗口的所有敌机被摧毁的动画，并且在动画播放完成后出现了新的敌机，如图所示。



5．实现敌机精灵的飞行

我们需要随机设置敌机的速度，实现让敌机沿垂直方向向下飞入游戏窗口的效果。如果敌机从下方飞出了游戏窗口，那么会被设置为初始状态。

① 修改reset_plane()方法，根据飞机的最大速度max_speed设置随机的初始速度。

② 重写父类的update()方法，实现敌机精灵的飞行功能。

③ 注释测试代码中测试修改游戏得分的代码。

11.6.3 派生敌机子类

5．实现敌机精灵的飞行

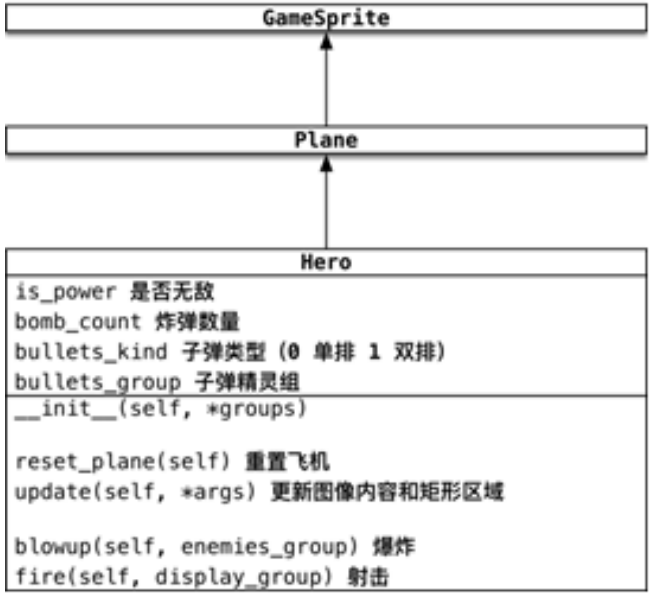
运行游戏，可以看到小敌机从游戏窗口的上方徐徐而来，且不停地出现新的小敌机，给人一种无穷无尽的感觉，如图所示。

11.6.4 派生英雄飞机子类

本小节将从Plane类派生一个表示英雄飞机的子类，该子类具有操控英雄飞机的功能，但不具备发射子弹消灭敌机的功能。

1．设计英雄飞机类

按照游戏说明设计英雄飞机Hero类，其类图如图所示：



11.6.4 派生英雄飞机子类

1．设计英雄飞机类

Hero类中包括is_power、bomb_count等几个属性，关于这几个属性的说明如表所示。

属性	说明
is_power	是否无敌。英雄飞机刚登场有3s的无敌时间
bomb_count	炸弹数量，默认携带 3 颗炸弹
bullets_kind	子弹类型，0表示单排；1表示双排
bullets_group	子弹精灵组。后续的碰撞检测需要使用

11.6.4 派生英雄飞机子类

1．设计英雄飞机类

Hero类不仅需要重写父类的方法，而且需要单独增加两个方法，关于这些方法的说明如表所示。

属性	说明
reset_plane()	重置飞机。英雄飞机被撞毁后，重置飞机的属性并发送自定义事件
update()	更新飞机。首先调用父类方法显示英雄飞机图像，然后根据 update() 方法的参数修改英雄飞机的位置，并将英雄飞机限定在游戏窗口之内
blowup()	引爆炸弹。炸毁游戏窗口内部的所有敌机，并返回得分
fire()	射击。创建 3 个子弹精灵并添加到子弹精灵组以及 display_group

11.6.4 派生英雄飞机子类

2．实现英雄类的基本功能

① 在game_items模块的顶部定义记录英雄默认炸弹数量的全局常量。

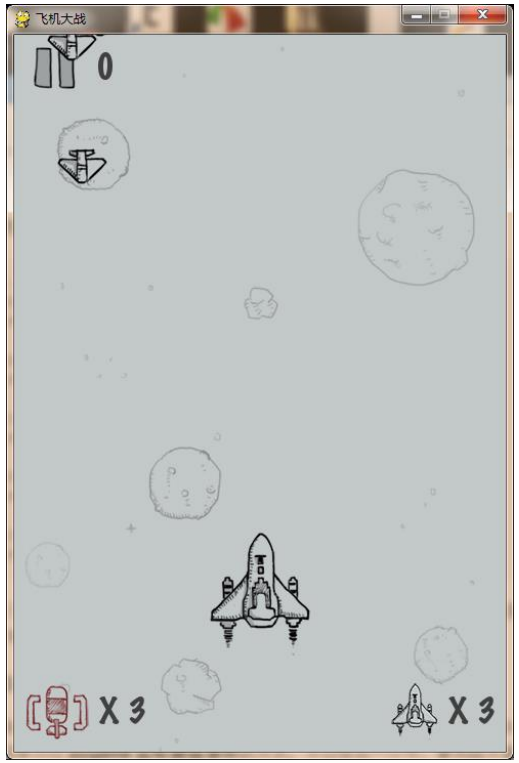
② 在game_items模块中定义Hero类。

③ 修改Game类的构造方法，将原有使用Plane类创建英雄飞机的代码替换为使用Hero类创建英雄飞机，并且使用英雄的炸弹数量属性设置指示器面板的显示。

11.6.4 派生英雄飞机子类

2．实现英雄类的基本功能

运行游戏，可以看到英雄飞机出现在游戏窗口中间靠下的位置，并且在指示器面板中显示的炸弹数量为3，如图所示。



11.6.4 派生英雄飞机子类

3．快速移动英雄飞机

下面实现使用键盘的方向键（↑、↓、←、→）控制英雄飞机在游戏窗口快速移动的功能。

第10章介绍了pygame监听键盘事件的方式，但这种方式需要用户抬起和按下方向键，不适合处理英雄飞机快速移动的需求。

因此，这里将为大家介绍和实现**持续按键的处理方式。**

11.6.4 派生英雄飞机子类

（1）持续按键的处理

pygame 专门针对持续按键的游戏开发需求提供了一种处理键盘的方式，即使用key模块提供的get_pressed()方法获得当前时刻的按键元组，然后使用按键常量作为元组索引，判断某一个键是否被按下，按下则对应的值是1，没按下则对应的值是0。

（2） 持续按键的方向判断

在飞机大战游戏中，英雄飞机的移动方向分为水平和垂直两种。那么，怎样编写代码才能简化我们对方向的判断呢？

使用keys[pygame.K_RIGHT] - keys[pygame.K_LEFT]计算水平移动的基数，代码如下：

```
move_hor = keys[pygame.K_RIGHT] - keys[pygame.K_LEFT]
```

使用以上代码计算水平方向移动的基数，其原理如表所示。

用户操作	keys[pygame.K_RIGHT]	keys[pygame.K_LEFT]	移动基数	结果
不按键	0	0	0	不移动
按下右键	1	0	1	向右移动1
按下左键	0	-1	-1	向左移动1
按下左右键	1	1	0	不移动

11.6.4 派生英雄飞机子类

（2）持续按键的方向判断

按照pygame坐标系的设定，x 轴沿水平方向向右，其数值逐渐增加。因此，这里可以先把移动基数值作为英雄飞机水平移动的基数（1表示向右移动，-1表示向左移动），之后用移动基数值乘以英雄飞机的速度，便可以计算得到水平方向需要移动的距离了。

英雄飞机垂直移动的基数可以采用同类似的方法进行计算。

11.6.4 派生英雄飞机子类

（3）移动英雄飞机

在 Hero 类中重写父类的update() 方法，在其中：

① 首先调用父类方法处理显示图像的更新；

② 然后增加修改飞机矩形区域的代码。

4．炸毁游戏窗口内部的敌机

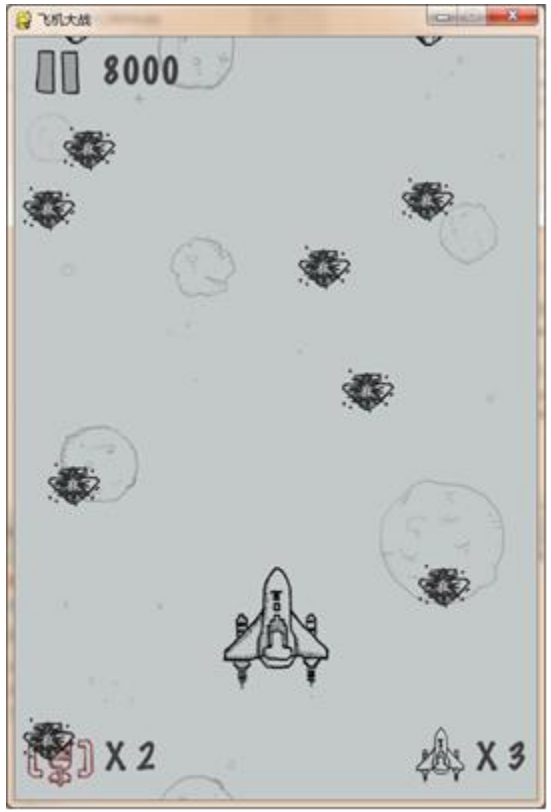
在Hero类中实现blowup()方法，将出现在游戏窗口内的敌机全部引爆，同时计算并返回得分。

修改Game类的event_handler()方法，删除监听到玩家按下字母b的事件处理中的测试代码，让英雄飞机调用blowup()方法，并且根据返回的分数做后续处理，包括更新游戏得分、更新炸弹数量显示以及判断是否升级到下一个关卡。

11.6.4 派生英雄飞机子类

4．炸毁游戏窗口内部的敌机

运行游戏，按下键盘的b键，可以看到游戏窗口中的所有敌机全部被炸毁，如图所示。



11.6.4 派生英雄飞机子类

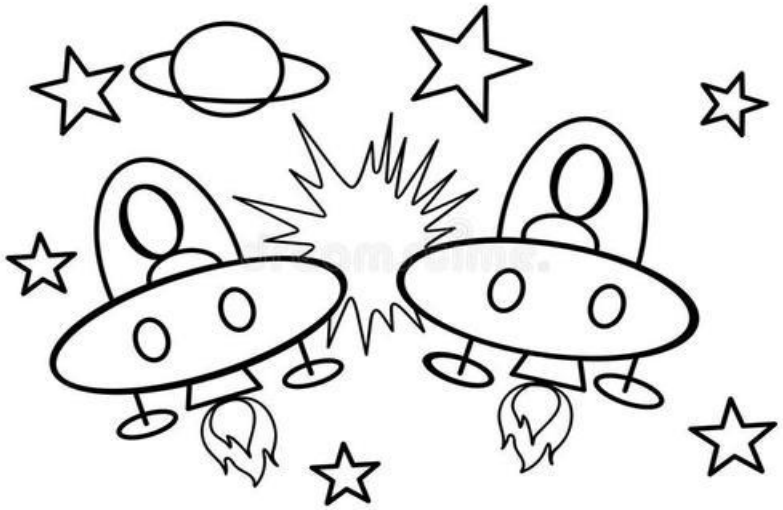
4．炸毁游戏窗口内部的敌机

另外，通过控制台的输出也可以发现，每次引爆的炸弹并不是炸毁敌机精灵组中的所有敌机，而是炸毁了出现在游戏窗口中的敌机。控制台的输出结果如下：

```
炸毁了 8 架敌机，得分 8000
炸毁了 13 架敌机，得分 13000
炸毁了 15 架敌机，得分 15000
```

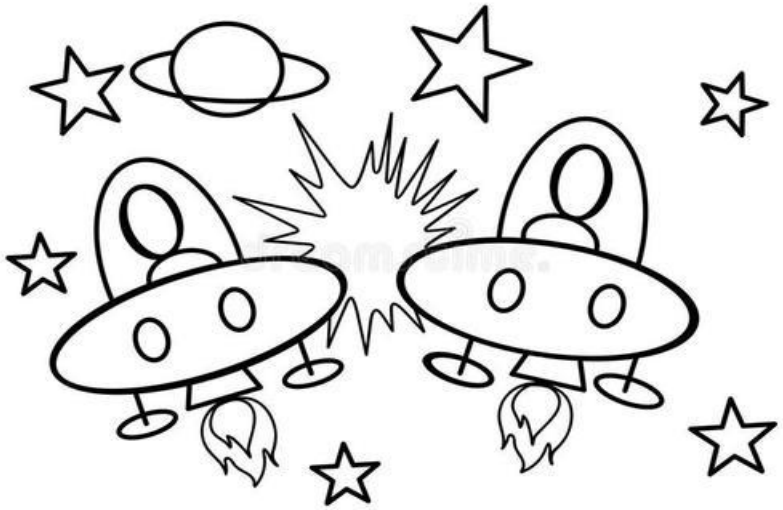
目录页

11.7 碰撞检测



碰撞检测是指在每一次游戏循环执行时检测游戏精灵之间是否发生碰撞，例如，敌机碰到英雄飞机、子弹碰到敌机等。碰撞检测在游戏开发中是至关重要的，它直接关系着玩家的游戏体验。

11.7.1 碰撞检测的实现



pygame的sprite模块中提供了实现碰撞检测功能的相关方法，使用这些方法可以实现碰撞检测。sprite模块还可以配合mask模块实现高质量的碰撞检测。下面先为大家介绍sprite模块的碰撞检测方法，再分别实现碰撞检测和高质量的碰撞检测。

11.7.1 碰撞检测的实现

1．碰撞检测方法

sprite模块中提供了两个碰撞检测的方法：spritecollide()和groupcollide()，关于这两个方法的介绍如下。

（1） spritecollide()方法

spritecollide()方法用于检测某个精灵是否和某个精灵组中的精灵发生碰撞，其语法格式如下所示：

```
spritecollide(sprite, group, dokill, collided = None) -> Sprite_list
```

以上方法中各参数的含义如下。

- sprite：表示要检测的精灵。
- group：表示要检测的精灵组。
- dokill：表示是否移除，若为True，会在检测到碰撞后移除group中与 sprite发生碰撞的精灵。

11.7.1 碰撞检测的实现

1．碰撞检测方法

sprite模块中提供了两个碰撞检测的方法：spritecollide()和groupcollide()，关于这两个方法的介绍如下。

（1） spritecollide()方法

spritecollide()方法用于检测某个精灵是否和某个精灵组中的精灵发生碰撞，其语法格式如下所示：

```
spritecollide(sprite, group, dokill, collided = None) -> Sprite_list
```

以上方法中各参数的含义如下。

- collided：表示用来检测碰撞的函数，若传入None，则使用精灵的rect属性来判断是否发生碰撞。

spritecollide()方法会返回group中与sprite发生碰撞的所有精灵的列表。

11.7.1 碰撞检测的实现

1．碰撞检测方法

sprite模块中提供了两个碰撞检测的方法：spritecollide()和groupcollide()，关于这两个方法的介绍如下。

（2） groupcollide()方法

groupcollide()方法用于检测两个精灵组之间是否有精灵发生碰撞，其语法格式如下所示：

```
groupcollide(group1, group2, dokill1, dokill2, collided = None) -> Sprite_dict
```

以上方法中各参数的含义如下。

- group1：表示要检测的精灵组1。
- group2：表示要检测的精灵组2。
- dokill1：表示是否从精灵组1移除，如果为True，会将发生碰撞的精灵从group1移除。

11.7.1 碰撞检测的实现

1．碰撞检测方法

sprite模块中提供了两个碰撞检测的方法：spritecollide()和groupcollide()，关于这两个方法的介绍如下。

（2） groupcollide()方法

groupcollide()方法用于检测两个精灵组之间是否有精灵发生碰撞，其语法格式如下所示：

```
groupcollide(group1, group2, dokill1, dokill2, collided = None) -> Sprite_dict
```

以上方法中各参数的含义如下。

- dokill2：表示是否从精灵组2移除，如果为True，会将发生碰撞的精灵从group2移除。
- collided：表示用来检测碰撞的函数，如果传入None，使用精灵的rect属性来判断是否发生碰撞。

groupcollide()方法会返回一个字典，该字典的键为group1中检测到被碰撞的精灵，值为 group2中与key发生碰撞的所有精灵的列表。

11.7.1 碰撞检测的实现

2．碰撞检测

① 在Game类构造方法的末尾增加测试代码，将所有创建的敌机对象都停止到屏幕中，方便观察碰撞检测的执行效果。

② 在Game类中实现一个专门负责碰撞检测的方法check_collide()。

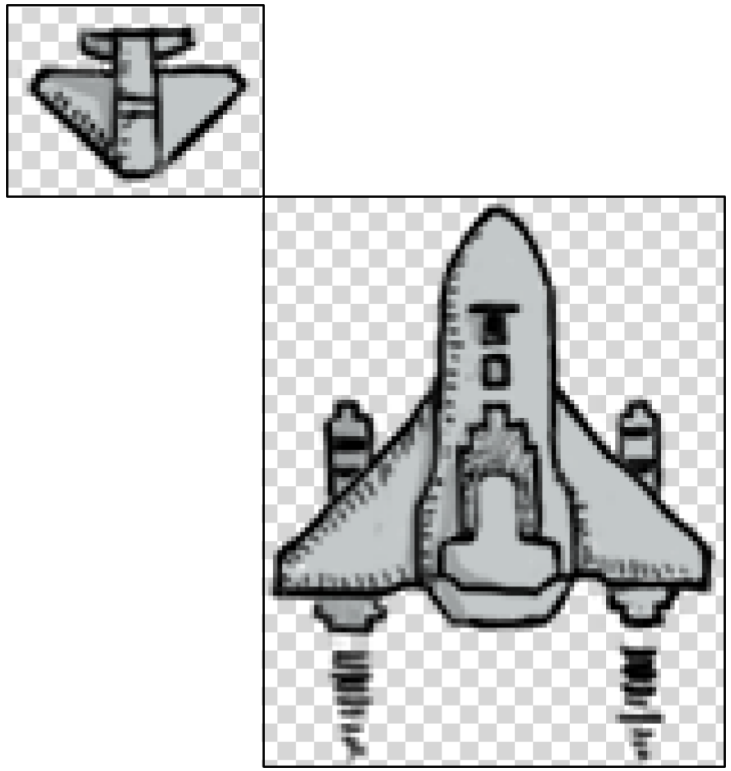
③ 在游戏循环中恢复指示器面板的代码之后，调用刚刚实现的check_collide()方法。

11.7.1 碰撞检测的实现

2．碰撞检测

运行游戏，通过方向键慢慢地让英雄飞机靠近敌机，可以发现英雄飞机即将靠近敌机时会撞毁敌机，而不是两架飞机真正地碰撞在一起后才出现撞毁效果。

之所以出现以上情况，是因为spritecollide()方法的collided参数为None时该函数使用精灵的rect属性来判断是否发生碰撞，此时只要两个精灵的矩形区域发生重叠，就认为精灵之间发生了碰撞，如图所示。



11.7.1 碰撞检测的实现

3．高品质的碰撞检测

前面通过spritecollide()方法实现的碰撞检测不够精确，此时可以给该方法传入一个pygame.sprite.collide_mask参数，只检测精灵图像中有颜色的区域。

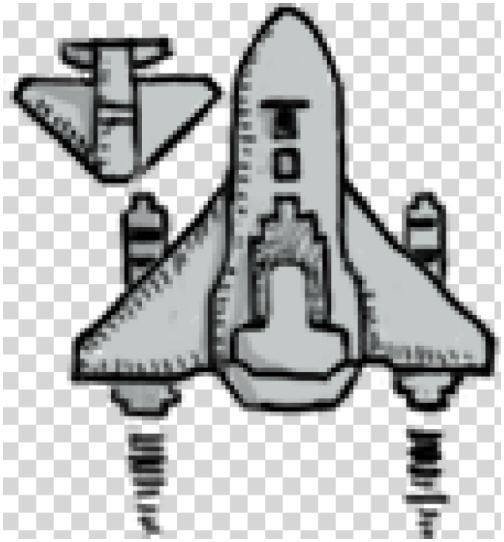
修改前面完成的check_collide()方法，验证能否实现高品质的碰撞检测。

11.7.1 碰撞检测的实现

3.高品质的碰撞检测

如果程序需要频繁地碰撞检测，那么可以在创建精灵时为精灵添加一个遮罩，也就是添加mask属性，以提升程序的执行性能。遮罩可以理解为图像的轮廓填充，也就是先为图像描边再填色。

在做碰撞检测时，图像中有颜色的部分会被认为是精灵的实体部分，没有颜色的部分会被忽略，如图所示。



11.7.1 碰撞检测的实现

3．高品质的碰撞检测

前面通过spritecollide()方法实现的碰撞检测不够精确，此时可以给该方法传入一个pygame.sprite.collide_mask参数，只检测精灵图像中有颜色的区域。

修改前面完成的check_collide()方法，验证能否实现高品质的碰撞检测。

在GameSprite类构造方法的末尾给精灵添加mask属性。

11.7.2 敌机撞毁英雄

如果敌机在飞行途中与英雄飞机相撞，那么会撞毁英雄飞机。

如果英雄飞机生命值不为0，那么英雄飞机会重新出现在撞毁的位置继续战斗；如果英雄飞机生命值为0，那么游戏结束。

- 撞毁英雄飞机的敌机同样要播放被撞毁动画。
- 动画播放完成后，敌机被设置为初始状态，会再次从游戏窗口上方飞入屏幕进行战斗。

11.7.2 敌机撞毁英雄

1．英雄被撞毁

- 首先，删除之前在Game类构造方法末尾增加的测试代码，让敌机恢复原有的飞行状态。
- 然后，修改Game类的check_collide()方法：在check_collide()方法中判断英雄飞机是否处于无敌状态，若处于无敌状态则不做碰撞检测，否则检测英雄飞机和所有敌机的碰撞。如果英雄飞机撞到敌机，那么英雄飞机会被撞毁，撞毁它的敌机会被撞毁。

此时的游戏存在一个问题：若英雄飞机与敌机距离较近，引爆炸弹摧毁敌机后英雄飞机可能会被敌机的残骸撞毁。若不希望英雄飞机被敌机残骸撞毁，可以过滤碰撞检测之后的列表，过滤出已经被撞毁的敌机。

11.7.2 敌机撞毁英雄

2．发布英雄飞机牺牲通知

按照游戏说明，在英雄飞机牺牲的动画播放完成之后，新的英雄飞机才能登场，游戏画面才会更新。

每当飞机的被摧毁动画播放完成之后，都会调用自己的reset_plane()方法来重设飞机的数据，但不能在Hero类的代码中直接更新游戏的画面。

要解决上述问题，需要使用event模块的post()方法发布一个英雄牺牲的用户自定义事件，如此便可以在Game类的事件监听方法中监听事件，并实现游戏画面的更新。

11.7.2 敌机撞毁英雄

2．发布英雄飞机牺牲通知

按照游戏说明，在英雄飞机牺牲的动画播放完成之后，新的英雄飞机才能登场，游戏画面才会更新。

每当飞机的被摧毁动画播放完成之后，都会调用自己的reset_plane()方法来重设飞机的数据，但不能在Hero类的代码中直接更新游戏的画面。

要解决上述问题，需要使用event模块的post()方法发布一个英雄牺牲的用户自定义事件，如此便可以在Game类的事件监听方法中监听事件，并实现游戏画面的更新。

- 在game_items模块的顶部定义记录英雄牺牲事件代号的全局常量
- 在Hero类中重写父类的reset_plane()方法
- 修改Game类的event_handler()方法，增加英雄牺牲事件的监听，一旦监听到英雄牺牲，需要修改生命计数以及更新生命计数和炸弹数量

11.7.2 敌机撞毁英雄

3．设置英雄无敌时间

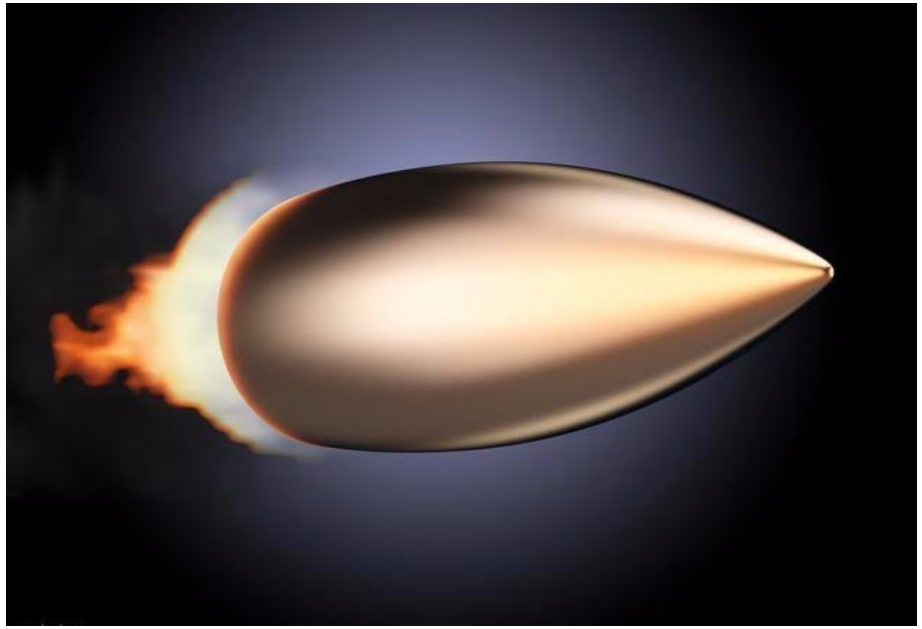
当屏幕上出现较多的敌机时，因为目前的程序没有设定英雄飞机的无敌状态，所以英雄飞机再次登场后会立即被撞毁。下面我们为英雄飞机设置无敌时间。

每当飞机的被摧毁动画播放完成之后，都会调用自己的reset_plane()方法来重设飞机的数据，但不能在Hero类的代码中直接更新游戏的画面。

要解决上述问题，需要使用event模块的post()方法发布一个英雄牺牲的用户自定义事件，如此便可以在Game类的事件监听方法中监听事件，并实现游戏画面的更新。

- 在game_items 模块的顶部定义记录取消英雄无敌事件的全局常量
- 在Hero类的reset_plane()方法末尾设定定时器事件
- 修改之前代码中英雄飞机的无敌标记，将self.is_power的值设置为True
- 修改Game类的event_handler()方法，增加取消英雄无敌事件的监听，一旦监听到取消英雄无敌，就需要修改英雄的无敌属性并且关闭定时器

11.7.3 英雄发射子弹



本小节主要完成英雄飞机发射子弹的操作，包括设计子弹类、实现英雄发射子弹的功能以及实现子弹击中敌机的功能。

11.7.3 英雄发射子弹

1．设计子弹类

按照游戏说明设计一个继承GameSprite的子弹类Bullet，Bullet类的类图如图所示。

11.7.3 英雄发射子弹

2．实现英雄发射子弹的功能

① 在game_items模块的顶部定义记录英雄发射子弹的定时器事件的全局常量

② 在Hero类的构造方法的末尾设置英雄发射子弹的定时器事件

③ 在Hero类中实现fire()方法

④ 修改Game类的event_handler()方法，增加英雄发射子弹事件的监听

11.7.3 英雄发射子弹

3．实现子弹击中敌机的功能

下面扩展Game类的check_collide()方法，实现子弹击中并摧毁敌机的功能。

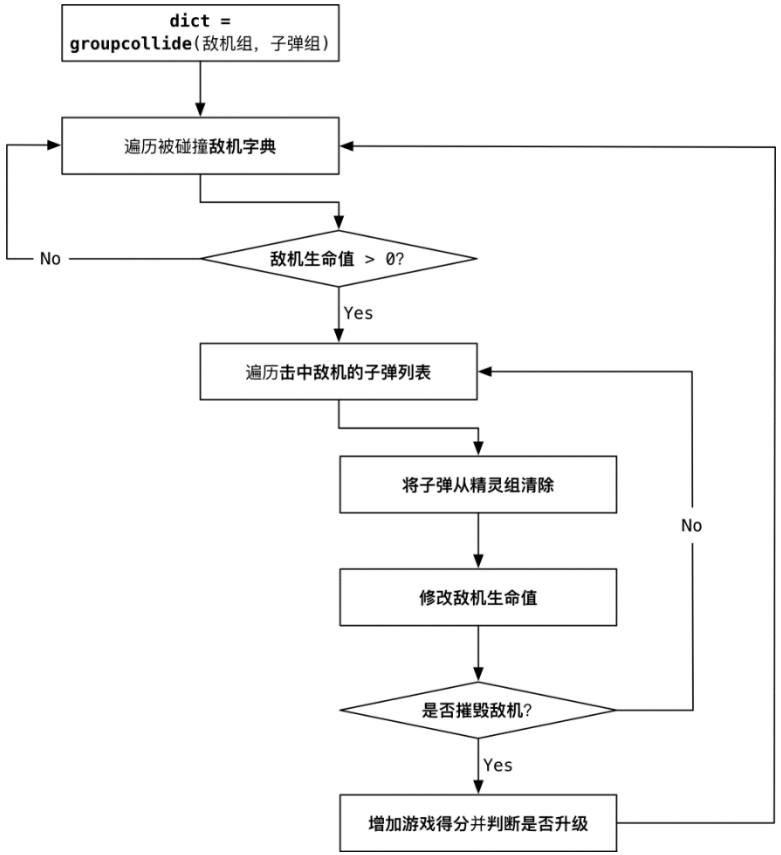
sprite模块spritecollide()方法可以便捷地实现一对多关系的碰撞检测，即一架英雄飞机与多架敌机。但此时需要处理多对多关系的碰撞检测，即多发子弹对多架敌机。针对这种需求，sprite模块提供了另外一个方法groupcollide()。

使用groupcollide()方法能够方便地检测敌机精灵组和子弹精灵组中的精灵是否发生了碰撞。若检测到碰撞会返回一个字典，字典的key为一个精灵组中检测到被碰撞的精灵，value为另一个精灵组中与key发生碰撞的所有精灵的列表。

11.7.3 英雄发射子弹

3．实现子弹击中敌机的功能

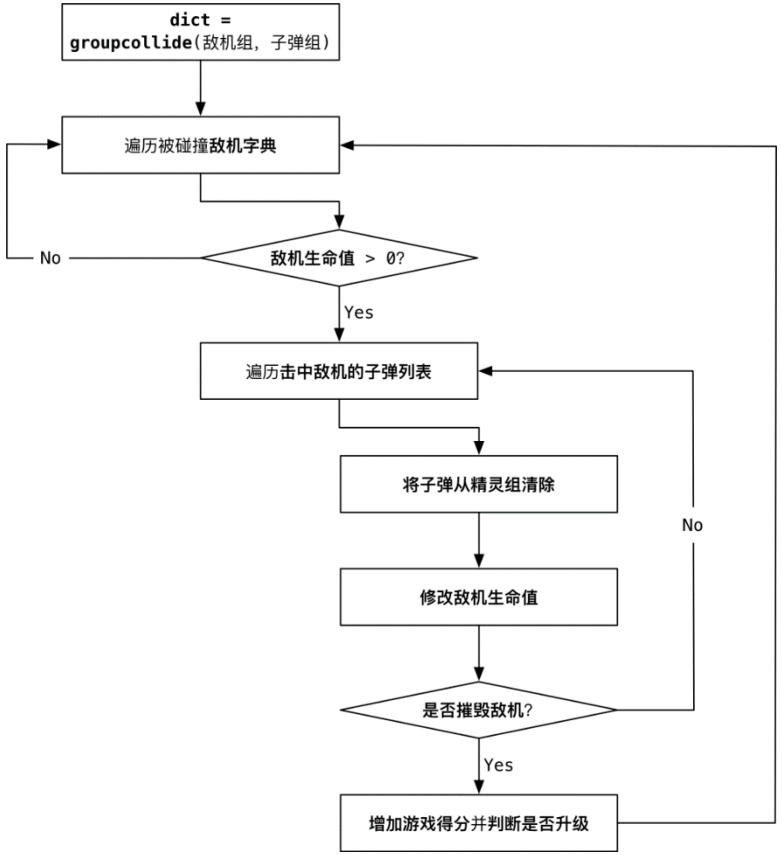
子弹击中敌机的流程，如图所示。



11.7.3 英雄发射子弹

3．实现子弹击中敌机的功能

子弹击中敌机的流程，如图所示。



在Game类的check_collide()方法的末尾增加处理子弹击中敌机的代码。

运行游戏，可以看到英雄飞机能发射子弹了，但是，当英雄飞机闯到第三关牺牲之后，玩家按下空格键开启新游戏，新游戏中仍然是第三关配置的敌机，而不是第一关的小敌机。

修改Game类的reset_game()方法，在reset_game()方法末尾先清空所有的敌机和英雄飞机残留的子弹，之后重新创建敌机对象。

11.7.4 英雄拾取道具

游戏开始后，道具每隔30秒会从游戏窗口上方的随机位置飞出，包括炸弹补给和子弹增强道具。下面实现英雄拾取道具的功能。

11.7.4 英雄拾取道具

1.设计道具类

在设计道具类之前，我们需要明确程序中道具位置的处理方式，如图所示。

为了避免在程序中反复创建相同的道具精灵，可以按照以下思路进行处理。

（1）在游戏初始化时创建两个道具精灵，并将道具精灵的初始位置设为游戏窗口的下方。

（2）游戏开始后，每隔30秒调用道具精灵的投放炸弹的方法，将道具精灵设置到游戏窗口上方的随机位置，准备开始垂直向下运动。

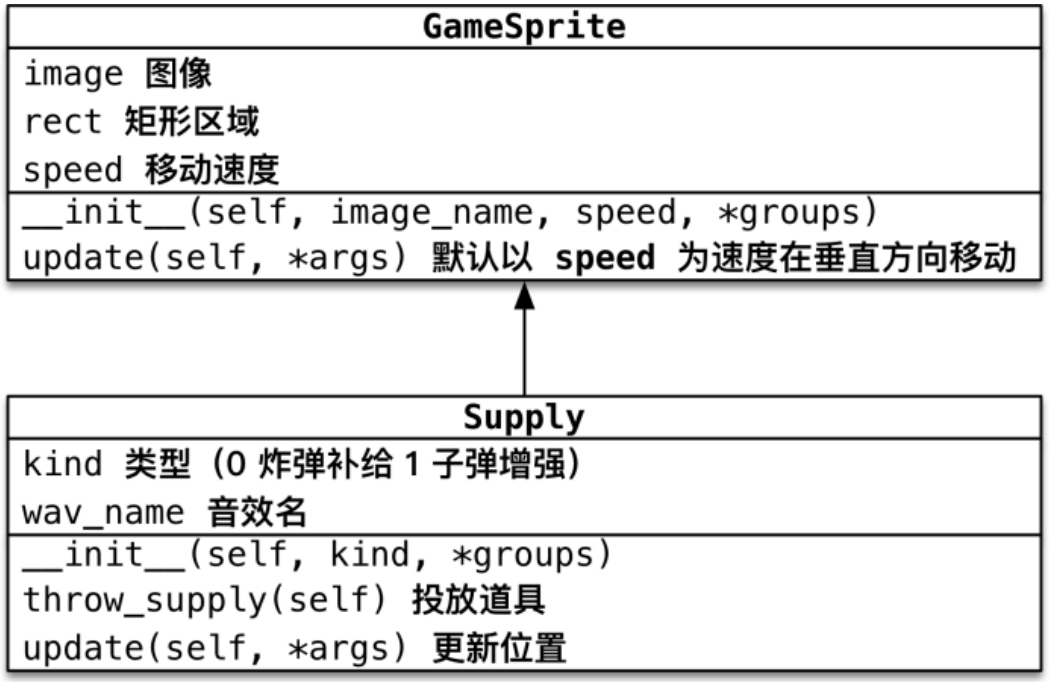
（3）若道具精灵向下方运动时遇到了英雄飞机，则设置相关属性，并且将道具精灵的位置设为游戏窗口的下方。

（4）道具精灵处于游戏窗口下方时不再更新位置。

11.7.4 英雄拾取道具

1.设计道具类

根据以上思路，设计了一个道具类Supply类，Supply类的类图如图所示。



11.7.4 英雄拾取道具

2.定时投放道具

下面利用定时器事件实现定时投放道具的功能。

① 在game_items模块的顶部定义记录投放道具的定时器事件的全局常量。

② 实现Game类的create_supplies()方法，在create_supplies()方法中创建两个道具精灵，并设置定时器事件。

③ 修改Game类的event_handler()方法，增加投放道具事件的监听，一旦监听到投放道具事件，将从道具精灵组中随机取出一个道具，让道具调用throw_supply()方法开始投放。

11.7.4 英雄拾取道具

3.英雄拾取道具

下面对Game类的check_collide()方法进行扩展，实现英雄拾取道具的功能。

① 在game_items模块的顶部定义记录关闭子弹增强的定时器事件的全局常量。

② 在Game类的check_collide()方法末尾增加处理英雄拾取道具的代码。

③ 修改Game类的event_handler()方法，增加关闭子弹增强事件的监听，一旦监听到子弹增强事件，恢复子弹类型并且关闭定时器事件。

11.7.4 英雄拾取道具

3.英雄拾取道具

下面对Game类的check_collide()方法进行扩展，实现英雄拾取道具的功能。

运行游戏，可以看到英雄飞机成功地拾取了道具，游戏更新了拾取后的结果：拾取炸弹补给后，游戏窗口的炸弹数量加1；拾取子弹增强后，英雄飞机发射的子弹变成双排。

目录页

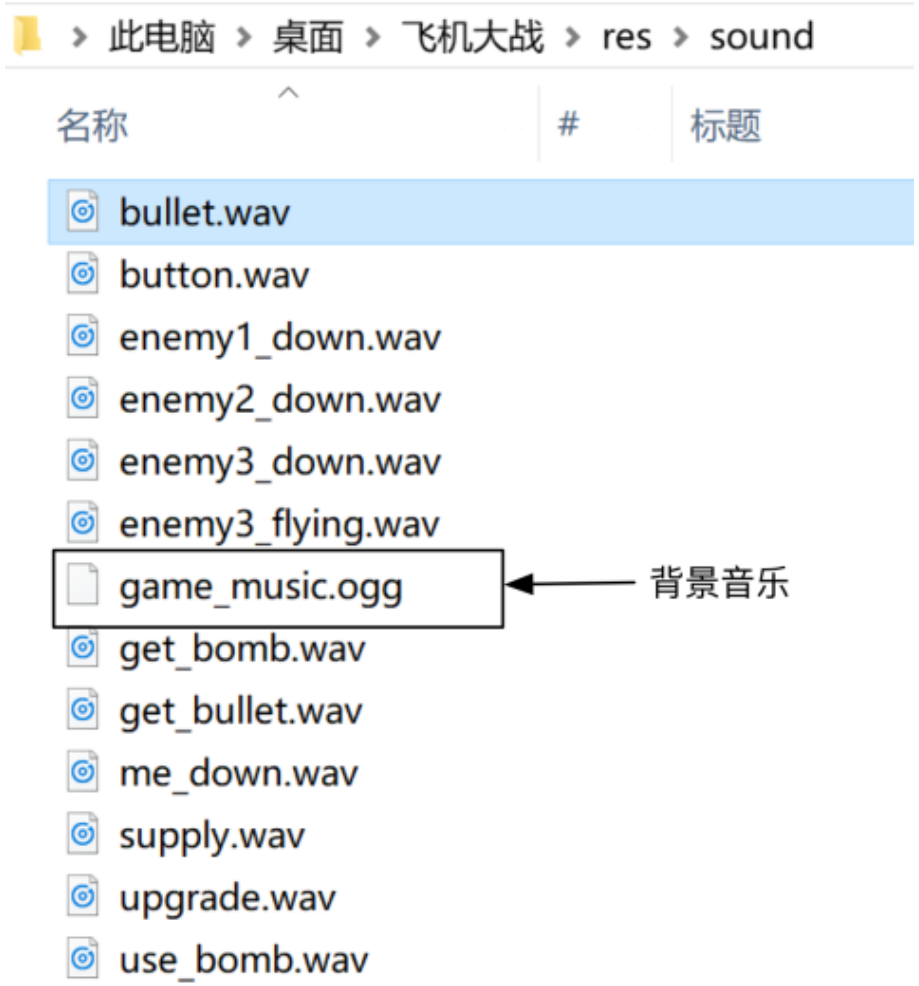
11.8 音乐和音效

游戏又被称为第九种艺术，是各项艺术的集合体。在一个独立且完整的游戏世界中，音乐是不可或缺的因素。

11.8.1 音乐播放器类的设计

游戏只有一首循环播放的背景音乐，但可以有多首游戏音效。每个音效都是一个单独的对象，它会在游戏需要时播放。

飞机大战游戏需要使用的背景音乐和音效文件保存在 res/sound 目录下，打开的sound 目录如图所示。



11.8.1 音乐播放器类的设计

为了简化在游戏中对音乐和音效播放的控制，我们设计一个表示音乐播放器的类MusicPlayer。MusicPlayer类的类图如图所示。

MusicPlayer
sound_dict 音效字典
__init__(self, music_file) 加载背景音乐，创建音效对象字典 play_sound(self, wav_name) 播放音效 play_music() 播放背景音乐 pause_music(is_paused) 暂停/恢复背景音乐

11.8.1 音乐播放器类的设计

MusicPlayer类中还封装了多个方法，关于这些方法的说明如表所示。

方法	说明
__init__(self, music_file)	参数 music_file 表示背景音乐文件名。res/sound目录下的其他文件都是音效文件
play_sound()	播放游戏音效
play_music()	静态方法，播放背景音乐
pause_music()	静态方法，暂停/恢复背景音乐

11.8.2 加载和播放背景音乐

1. 在game_music模块文件的顶部导入需要使用的模块
2. 定义MusicPlayer类，并且实现加载及播放背景音乐相关的方法
3. 在Game类的构造方法的末尾删除之前的测试代码，创建音乐播放器并且循环播放背景音乐
4. 修改Game类的event_handler()方法，在监听到玩家按下空格键暂停或恢复游戏的同时，暂停或恢复游戏背景音乐的播放

11.8.3 加载和播放音效

1. 在MusicPlayer类的构造方法的末尾增加代码，从./res/sound目录加载所有的音频文件，并且将创建的声音对象添加到sound_dict音效字典中。
2. 在MusicPlayer类中实现播放音效的play_sound()方法。
3. 在Game类的指定位置逐一调用播放音效的方法。

① 发射子弹

② 引爆炸弹

③ 投放和拾取道具

④ 敌机爆炸和升级

⑤ 英雄爆炸

目录页

11.9 项目打包

我们可以利用Python的第三方库——pyinstaller对我们编写的游戏（但不仅限于游戏）项目进行打包。

11.9 项目打包

1．pyinstaller的安装和使用

由于pyinstaller库依赖其他模块，建议采用pip命令在线安装，而非离线安装包方式安装。安装命令如下：

```
pip install pyinstaller
```

切换至程序所在的目录，可以通过pyinstaller命令打包程序，具体命令如下：

```
pyinstaller 选项 Python源文件
```

以上命令的Python源文件表示程序的入口文件；选项表示一些控制生成软件包的辅助命令。

11.9 项目打包

1．pyinstaller的安装和使用

pyinstaller支持的常用选项如表所示。

选项	说明
-F/--onefile	将项目打包为单个可执行程序文件
-D/--onedir	将项目打包为一个包含可执行程序的目录（包含多个文件）
-d/--debug	将项目打包为debug版本的可执行文件
-w/--windowed/--noconsolc	指定程序运行时不显示命令行窗口（仅对Windows有效）
-c/--nowindowed/--console	指定使用命令行窗口运行程序（仅对Windows有效）
-o DIR/--out=DIR	指定spec文件的生成目录。若没有指定，则默认为当前目录

11.9 项目打包

1．pyinstaller的安装和使用

“pyinstaller 选项 Python源文件”命令执行完成后，可以看到源文件所在的目录中增加了两个目录：build和dist。其中：

- build目录是存储临时文件的目录，可以安全地删除；
- dist目录中包含一个与源文件同名的目录，该同名目录中包含了可执行的软件，以及可执行文件的动态链接库。

11.9 项目打包

2．使用pyinstaller打包游戏项目

在飞机大战目录中打开命令行窗口，使用pyinstaller命令打包程序的入门文件game.py：

```
pyinstaller -Fw game.py
```

名称	修改日期	类型	大小
Analysis-00.toc	2020/4/3 14:19	TOC 文件	119 KB
base_library.zip	2020/4/3 14:19	WinRAR ZIP 压缩...	762 KB
COLLECT-00.toc	2020/4/3 14:19	TOC 文件	98 KB
EXE-00.toc	2020/4/3 14:19	TOC 文件	3 KB
game.exe	2020/4/3 14:19	应用程序	2,249 KB
game.exe.manifest	2020/4/3 14:19	MANIFEST 文件	2 KB
PKG-00.pkg	2020/4/3 14:19	PKG 文件	2,010 KB
PKG-00.toc	2020/4/3 14:19	TOC 文件	2 KB
PYZ-00.pyz	2020/4/3 14:19	Python Zip Appli...	1,990 KB
PYZ-00.toc	2020/4/3 14:19	TOC 文件	21 KB

- ✓ 此时，大家只需要将game.exe文件和res目录置于同一目录中，双击game.exe便可启动游戏；
- ✓ 亦可将game.exe与res目录一同压缩，分享给其他人，在其他设备上运行游戏。

11.10 本章小结

本章围绕着面向对象的编程思想，分游戏简介、项目准备、游戏框架搭建、游戏背景和英雄飞机、指示器面板、逐帧动画和飞机类、碰撞检测、音乐和音效、项目打包这些部分开发和打包了一个具备完整功能的飞机大战游戏。通过对本章的学习，希望读者可以灵活地运用面向编程的编程技巧，并能将其运用到实际开发中。