# 项目：飞机大战

### 项目介绍

#### 1.1项目涉及前期课程知识点

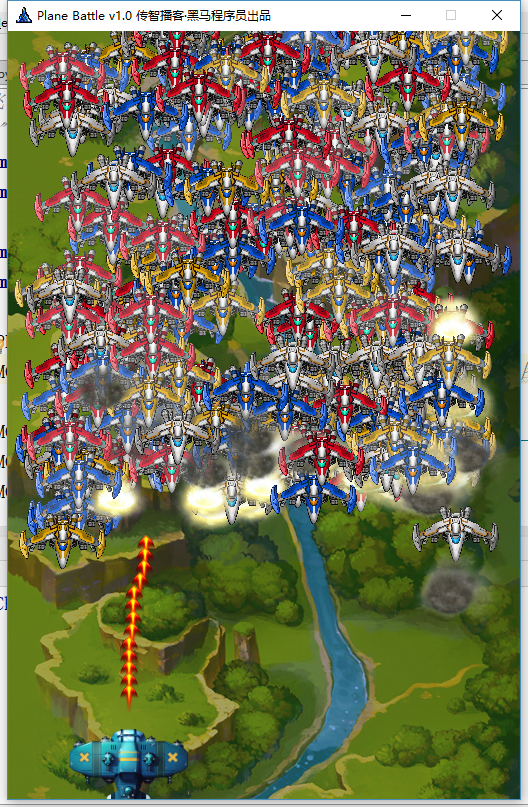
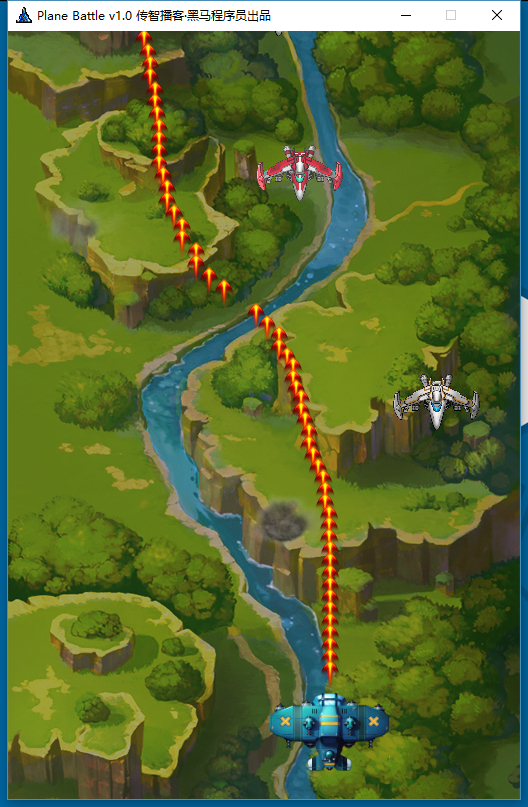
1.基础语法：（注释、变量、运算符、循环、分支、嵌套、函数、形参、实参、返回值、引用）

2.面向对象：（类、对象、成员变量、成员方法、魔术方法、构造方法、继承、重写、多态）

3.数据存储：（列表、元组、字典）

4.模块与包

#### 1.2项目介绍

#### 1.3主要功能介绍

1.背景卷轴式滚动

2.敌机自上而下俯冲式攻击

3.玩家发射子弹消灭敌机

4.敌机击中爆炸

### 创建项目

#### 2.1项目环境搭建

1.创建项目Battle（名称可以自定义）

2.创建资源保存目录（res）

3.拷贝资源文件到目录res

4.安装pygame

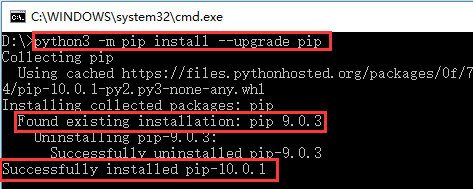
* 32位安装方式：

直接安装后拷贝到python安装目录中即可

* 64位安装方式：

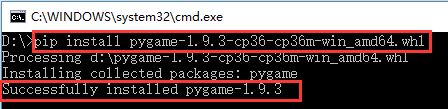
（1）更新pip安装工具

命令：**python -m pip install --upgrade pip**



（2）安装下载完成的pyganme安装包

命令:**pip install pygame-1.9.3-cp36-cp36m-win\_amd64.whl**



说明1：如果安装的版本不兼容，运行程序时会出现报错信息

说明2：安装下载完毕的安装包时，要注意执行命令的路径与安装包的路径在同一个目录下，例如：当前操作路径是在D盘根目录下

5.进入项目创建测试文件，导入pygame后运行，观察是否报错

（1）创建test.py文件

（2）输入import pygame

（3）运行文件

#### 2.2初始化项目模型

创建项目中需要使用的各种模型类

* 窗体 （pygame提供）
* 背景 （Background）
* 玩家飞机（PlayerPlane）
* 敌人飞机（EnemyPlane）
* 子弹 （Bullet）

### 窗体

#### 3.1程序结构初始化分析

在游戏（Game）类中定义运行程序的专用方法（run），对每个功能定义单独的管理方法，在run方法中调用每一个功能的管理方法，使整体结构按照功能进行划分，便于后期管理

#### 3.2窗体制作分析

1.创建窗体的大小由游戏的需求而定，本游戏使用的背景图片尺寸是512\*768，故窗体的大小设置为512\*768，单位像素

2.窗体显示在屏幕上的维持时间为1次，为了保障屏幕上显示的信息不被其他内容所遮盖，必须保障窗体反复显示在屏幕上，因此需要反复更新窗体显示的内容

3.相关文档

# 1.创建窗体对象  
pygame.display.set\_mode((width,height))  
# 参数：元组，包含窗体的宽，高  
# 返回值：窗体对象  
  
# 2.刷新界面显示窗体，该操作需要反复执行，否则屏幕上无法观察到动态刷新出的变化效果  
pygame.display.update()

#### 3.3代码实现1

import pygame # TODO 5.导入pygame模块

class Game:  
 # TODO 1.主程序，运行游戏入口  
 def run(self):  
 self.frame\_init() # TODO 4.执行窗体初始化  
 while True: # TODO 7.构造反复执行的机制，刷新窗体，使窗体保持在屏幕上  
 pygame.display.update() # TODO 8.刷新窗体  
 # TODO 3.初始化窗体  
 def frame\_init(self):  
 window = pygame.display.set\_mode((512, 768)) # TODO 6.初始化窗体

*# TODO 2.设置测试类入口操作*if \_\_name\_\_ == "\_\_main\_\_":  
 Game().run()

#### 3.4窗体设置分析

1.窗体标题栏显示的信息由图片和文字信息组成，对于界面操作有关的设置需要从pygame的文档中查询获取

2.相关文档

*# 3.加载图标文件为图片对象*pygame.image.load("图片路径")  
*# 参数：字符串，对应的图片路径  
# 返回值：图片对象  
  
# 4.设置窗体图标*pygame.display.set\_icon(image)  
*# 参数：图片对象  
  
# 5.设置窗体的标题文本*pygame.display.set\_caption("plane battle")  
*# 参数：字符串，窗体显示的文字*

#### 3.5代码实现2

class Game:  
 *# 初始化窗体* def frame\_init(self):  
 window = pygame.display.set\_mode((512, 768)) img = pygame.image.load("res/app.ico") *# TODO 1.加载图标文件为图片对象* pygame.display.set\_icon(img) *# TODO 2.设置窗体图标为图片* pygame.display.set\_caption("Plane Battle v1.0 传智播客·黑马程序员出品") *# TODO 3.设置窗体的标题*

#### 3.6关闭窗体分析

1.关闭窗体是在窗体中建立一种机制，保障时时刻刻监控用户的操作行为，这种机制称为事件监听机制

2.为窗体添加事件监听机制后要对产生的事件进行区分，并针对不同的事件完成对应的功能，对于界面操作有关的设置需要从pygame的文档中查询获取

3.关闭窗体仅仅是窗体消失，真正程序退出由系统指定完成，调用sys模块中的exit()操作即可完成

4.相关文档

# 6.获取pygame界面的所有事件  
pygame.event.get()  
# 返回值：事件列表  
  
# 7.获取事件类别  
event.type  
# QUIT none  
# ACTIVEEVENT gain, state  
# KEYDOWN unicode, key, mod  
# KEYUP key, mod  
# MOUSEMOTION pos, rel, buttons  
# MOUSEBUTTONUP pos, button  
# MOUSEBUTTONDOWN pos, button  
# JOYAXISMOTION joy, axis, value  
# JOYBALLMOTION joy, ball, rel  
# JOYHATMOTION joy, hat, value  
# JOYBUTTONUP joy, button  
# JOYBUTTONDOWN joy, button  
# VIDEORESIZE size, w, h  
# VIDEOEXPOSE none  
# USEREVENT code  
  
# 8.导入pygame本地设置的资源，包含程序执行过程中所使用的各种常量，例如键盘按钮，鼠标按钮等  
import pygame.locals  
  
# 9.窗体按钮X  
pygame.locals.QUIT  
  
# 10.退出系统  
sys.exit()

#### 3.7代码实现3

import sys *# TODO 6.导入系统模块*import pygame import pygame.locals *# TODO 5.导入pygame本地策略*

*# 游戏类*class Game:  
 def run(self):  
 self.frame\_init()   
 while True:   
 pygame.display.update()   
 self.event\_init() *# TODO 2.反复调用事件监听方法  
  
 # TODO 1.定义事件处理的方法* def event\_init(self):  
 for event in pygame.event.get(): *# TODO 3.获取当前发生的所有事件* if event.type == pygame.locals.QUIT : *# TODO 4.判断当前事件类别是不是点击窗体的关闭按钮* sys.exit() *# TODO 7.执行退出系统操作*

#### 3.8实用技巧

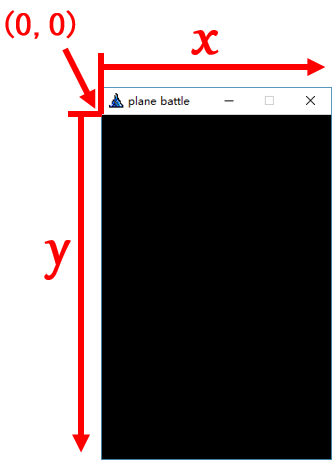
Python程序在进行外部资源模块导入时，通常按照下列顺序进行导入，中间使用空行分隔。如果模块中还包含有其他模块，按照从外到内的顺序依次导入。

* 系统资源模块
* 第三方资源模块
  + 主资源名称 例如：import pygame
  + 所属资源名称 例如: import pygame.locals
* 用户自定义开发模块

### 背景

#### 4.1背景图片加入窗体制作分析

1.背景图片加入窗体必须确认大小与位置信息，图片大小由原始素材文件决定（512\*768），位置必须放置在窗体的左上角（0，0）点

2.设置窗体对象的模型属性，包含图片，显示的x与y坐标

3.图片信息显示需要条用界面的操作，对于界面操作有关的设置需要从pygame的文档中查询获取

4.相关文档

# 11.添加要显示的对象到窗体中显示  
window.blit(img, (x, y))  
# 参数1：可显示的对象，对象可以是图片，可以是文本信息  
# 参数2：元组，包含显示的位置x与y坐标

#### 4.2代码实现1

*# 背景类*class Background:  
 *# TODO 1.制作背景构造方法，传入图片路径，x,y* def \_\_init\_\_(self, img\_path, x, y):  
 self.img = pygame.image.load(img\_path) *# TODO 2.背景图片* self.x = x *# TODO 2.窗体中放置的x坐标* self.y = y *# TODO 2.窗体中放置的y坐标*

class Game:  
def run(self):  
 self.frame\_init() self.model\_init() *# TODO 4.执行对象初始化* while True: pygame.display.update() self.event\_init()  *# TODO 3.初始化窗体中的对象* def model\_init(self):  
 background = Background("res/img\_bg\_level\_1.jpg",0,0) *# TODO 5.初始化背景对象，传入图片路径，放置在0,0位* self.window.blit(background.img, (background.x, background.y)) *# TODO 6.调用bilt方法，将图片加入到窗体中*

#### 4.3常量抽取

程序开发过程中，某些数据是一直保持不变的，对于此类数据通常设置为全局公共的变量，名称由全大写字母组成，中间使用下划线\_分隔

#### 4.4代码实现2

APP\_ICON = "res/app.ico" *# TODO 1.设计公共的图片常量，定义在文件中，方便管理*IMG\_BACKGROUND = "res/img\_bg\_level\_1.jpg" *# TODO 1.设计公共的图片常量，定义在文件中，方便管理*

class Game:  
 WINDOW\_WIDTH = 512 *# TODO 3.设计窗体尺寸常量，定义在类中，缩小作用范围* WINDOW\_HEIGHT = 768 *# TODO 3.设计窗体尺寸常量，定义在类中，缩小作用范围* def frame\_init(self):  
 self.window = pygame.display.set\_mode((Game.WINDOW\_WIDTH, Game.WINDOW\_HEIGHT)) *# TODO 4.使用窗体常量替换原始格式 # 初始化窗体* img = pygame.image.load(APP\_ICON) *# TODO 2.使用图片常量替换原始格式 # 加载图标文件为图片对象* pygame.display.set\_icon(img)   
 pygame.display.set\_caption("Plane Battle v1.0 传智播客·黑马程序员出品")   
  
 def model\_init(self):  
 background = Background(IMG\_BACKGROUND,0,0) *# TODO 2.使用图片常量替换原始格式 # 初始化背景对象，传入图片路径，放置在0,0位* self.window.blit(background.img, (background.x, background.y))

#### 4.5抽取模型类分析

1.程序制作过程中，对于具有相同或相似功能的类，建议抽取其共性功能，并通过继承机制实现通用模型的开发。本项目中所有的实体模型类，例如背景、飞机、子弹等都具有图片、横坐标、纵坐标、显示等属性和方法，因此抽取通用模型类

2.抽取通用构造方法，包含图片、横坐标、纵坐标

3.抽取通用方法，装载图片到界面中

#### 4.6代码实现3

*# TODO 2.创建所有显示的图形父类Model*class Model:  
 window = None *# TODO 7.定义主窗体对象，用于模型访问使用  
 # TODO 3.构造方法* def \_\_init\_\_(self,img\_path,x,y):  
 self.img = pygame.image.load(img\_path) *# TODO 4.背景图片* self.x = x *# TODO 4.窗体中放置的x坐标* self.y = y *# TODO 4.窗体中放置的y坐标  
  
 # TODO 5.将模型加入窗体的方法抽取到父类* def display(self):  
 Model.window.blit(self.img, (self.x, self.y)) *# TODO 8.使用Model的类变量访问窗体对象 # 调用bilt方法，将图片加入到窗体中*

class Background(Model):  
 """ TODO 1.将操作抽取到父类  
 def \_\_init\_\_(self,img\_path,x,y):  
 self.img = pygame.image.load(img\_path)   
 self.x = x   
 self.y = y   
 """

*# 玩家类*class PlayerPlane(Model):  
 pass  
*# 敌机类*class EnemyPlane(Model):  
 pass  
*# 子弹类*class Bullet(Model):  
 pass

class Game:  
 def frame\_init(self):  
 self.window = pygame.display.set\_mode((Game.WINDOW\_WIDTH, Game.WINDOW\_HEIGHT))  
 Model.window = self.window *# TODO 9.将窗体对象传入模型类中* img = pygame.image.load(APP\_ICON)  
 pygame.display.set\_icon(img)  
 pygame.display.set\_caption("Plane Battle v1.0 传智播客·黑马程序员出品")  
  
 def model\_init(self):  
 background = Background(IMG\_BACKGROUND,0,0)  
 background.display() *# TODO 6.使用抽取的Model类中的display方法完成* """ TODO 6.使用公共方法完成  
 self.window.blit(background.img, (background.x, background.y))   
 """

#### 4.7单个背景移动分析

1.背景移动的操作实质是更改图片的位置后，并显示该图片。由于原始图片移动后的影像还存留在窗体中，因此看到的效果是有一部分内容是固定的显示内容，没有变化

2.制作向下的移动效果就是更改图片的y坐标，每次加固定值，值的大小决定了移动的速度

3.移动后更新操作是必须进行的否则仅进行数据变化，屏幕无显示效果

#### 4.8代码实现3

class Background(Model):  
 *# TODO 1.定义背景移动的方法* def move(self):  
 self.y += 1 *# TODO 2.纵坐标自增1*

class Game:  
 def run(self):  
 self.frame\_init()  
 self.model\_init()  
 while True:  
 self.background.move() *# TODO 4.调用背景移动操作，构造背景图片向下移动的效果* self.background.display() *# TODO 5.移动完毕后，将移动后的图片加入到窗体中* pygame.display.update()  
 self.event\_init()  
  
 def model\_init(self):  
 self.background = Background(IMG\_BACKGROUND,0,0) *# TODO 3.扩大背景对象的使用范围*

""" TODO 6.删除单独的将背景图片加入窗体的操作（不影响结果）  
 self.background.display() # TODO 3.更新调用方式   
 """

#### 4.9单个背景反复移动分析

1.背景移动到超过屏幕显示区域后，图片的显示效果已经无法观察到，此时需要将图片的位置重置到原始位置即可制作出反复移动的效果



2.重置原始位置就是将图片的纵坐标y设置成图片最初的纵坐标值0，设置的条件是当图片的位置超出了屏幕的显示范围，即y的值超过了768

#### 4.10代码实现4

class Background(Model):  
 def move(self):  
 # TODO 1.加入背景移动的情况判定  
 if self.y <= Game.WINDOW\_HEIGHT: # TODO 2.如果没有超出屏幕就正常移动  
 self.y += 1   
 else: # TODO 3.如果超出屏幕，恢复图片位置为原始位置  
 self.y = 0 # TODO 4.纵坐标 = 0

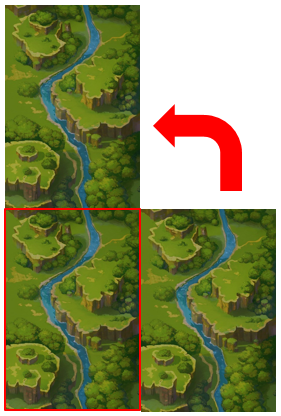
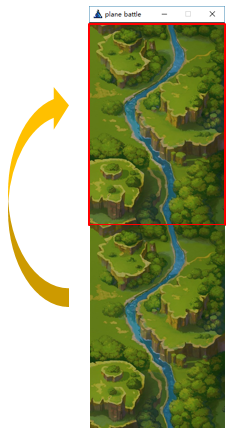
#### 4.11添加辅助背景分析

1.连续的背景移动不是构造超大的图片，而是将两张图片通过拼接的形式进行连接，构造出连续背景移动的效果

2.添加的背景图片称为辅助背景，与原始背景组合在一起构造出连续的移动效果

3.辅助背景无需重新创建新的对象，使用原始背景图片进行多次显示即可，只需要更改显示的位置

4.辅助背景必须与原始背景进行同步的移动操作，否则整体画面会有跳跃的感觉

#### 4.12代码实现5

class Background(Model):  
 # TODO 1.覆盖父类display方法，制作原始背景贴图+辅助背景贴图  
 def display(self):  
 Model.window.blit(self.img, (self.x, self.y)) # TODO 2.原始背景贴图，推荐使用super().display()  
 Model.window.blit(self.img, (self.x, self.y - Game.WINDOW\_HEIGHT)) # TODO 3.辅助背景，坐标位置与原始背景贴图上下拼接吻合

### 敌机

#### 5.1一架敌机制作分析

敌机的制作与背景的制作完全相同，区别之处是图片不同和贴图位置不同，模仿背景的制作完成敌机的加入即可。飞机默认进入位置先暂定为200,200处

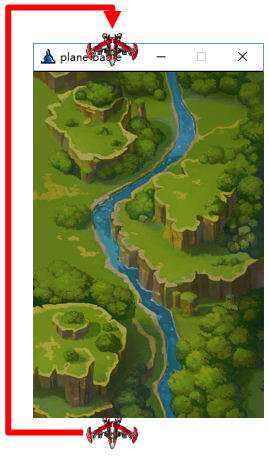
#### 5.2代码实现1

IMG\_ENEMY = "res/img-plane\_5.png" *# TODO 1.初始化敌机图片*

class Game:  
 def run(self):  
 self.frame\_init()   
 self.model\_init()   
 while True:   
 self.background.move()   
 self.background.display()   
 self.enemy.display() *# TODO 3.敌机显示到窗体中* pygame.display.update()   
 self.event\_init()   
  
 def model\_init(self):  
 self.background = Background(IMG\_BACKGROUND,0,0)   
 self.enemy = EnemyPlane(IMG\_ENEMY,200,200) *# TODO 2.初始化敌机对象，放置在200,200坐标处（临时）*

#### 5.3敌机移动制作分析

1.敌机的移动与背景的移动制作方式完全相同，区别之处是飞机的移动速度与移动完毕后的贴图位置与背景不同



2.飞机移动速度建议设置为4，比背景移动速度稍快

3.飞机飞行到窗体外部后，恢复y的值为窗体顶部，为了保障飞机恢复后不直接显示到屏幕中，坐标值建议设置为负值，至少超过飞机的高度

#### 5.4代码实现2

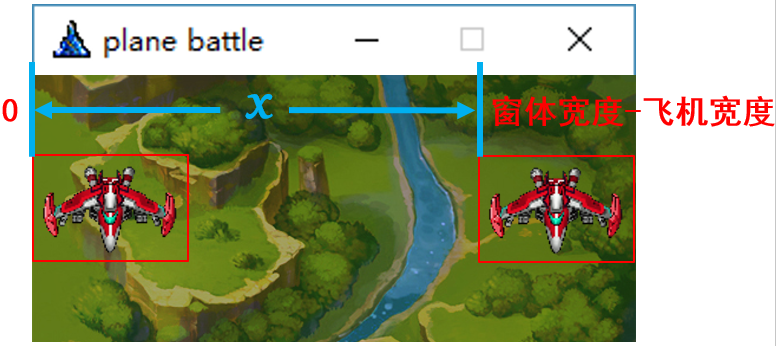
class EnemyPlane(Model):  
 # TODO 1.定义敌机移动的方法  
 def move(self):  
 # TODO 2.判断敌机是否超出屏幕  
 if self.y > Game.WINDOW\_HEIGHT: # TODO 3.如果超出屏幕  
 self.y = -68 # TODO 4.敌机纵坐标设置为-68，返回屏幕顶端  
 else: # TODO 5.敌机未超出屏幕  
 self.y += 4 # TODO 6.控制敌机向下移动，移动速度设置为4

class Game:  
 def run(self):  
 self.frame\_init()   
 self.model\_init()   
 while True:   
 self.background.move() self.background.display()   
 self.enemy.move() *# TODO 7.调用敌机移动操作* self.enemy.display()   
 pygame.display.update()   
 self.event\_init()

#### 5.5创建多驾敌机并水平分布制作分析

1.多驾敌机的保存使用列表完成

2.敌机生成的位置横坐标使用随机数生成，位置x从0到屏幕宽度-飞机宽度



#### 5.6代码实现3

import sys  
import random *# TODO 4.导入随机数模块*import pygame  
import pygame.locals

class Game:  
 def run(self):  
 self.frame\_init()  
 self.model\_init()  
 while True:  
 self.background.move()  
 self.background.display()  
 """ TODO 6.删除一架敌机的移动控制  
 self.enemy.move()   
 self.enemy.display()   
 """  
 for enemy in self.enemys: *# TODO 7.加入所有敌机* enemy.move() *# TODO 7.每驾敌机移动* enemy.display() *# TODO 7.每驾敌机加入窗体* pygame.display.update()  
 self.event\_init()  
  
 def model\_init(self):  
 self.background = Background(IMG\_BACKGROUND,0,0)  
 """ TODO 1.删除创建一个敌机对象  
 self.enemy = EnemyPlane(IMG\_ENEMY,200,200)   
 """  
 self.enemys = [] *# TODO 2.定义用于保存敌机的空列表，保存多驾敌机* for \_ in range(5) : *# TODO 3.循环产生5架敌机* self.enemys.append(EnemyPlane(IMG\_ENEMY,random.randint(0, Game.WINDOW\_WIDTH - 100 ),200)) *# TODO 5.向列表添加敌机，初始位置x为横向随机，纵向200*

#### 5.7多驾敌机垂直分布制作分析

1.敌机生成位置要进行垂直分布的随机设置，位置y从-屏幕高度到-飞机高度

2.敌机一旦超出屏幕，返回顶端时的横坐标和纵坐标也使用随机数进行控制，保障屏幕上的敌机飞行无规律

#### 5.8代码实现4

class EnemyPlane(Model):  
 *# TODO 1.覆盖init方法* def \_\_init\_\_(self):  
 self.img = pygame.image.load(IMG\_ENEMY) *# TODO 2.使用常量* self.x = random.randint(0,Game.WINDOW\_WIDTH - 100) *# TODO 3.设置x坐标随机生成 横向位置 0 到 屏幕宽度 - 飞机宽度(100)* self.y = random.randint( -Game.WINDOW\_HEIGHT, -68) *# TODO 4.设置y坐标随机生成 纵向位置 -屏幕高度 到 -飞机高度* def move(self):  
 if self.y > Game.WINDOW\_HEIGHT:  
 self.x = random.randint(0, Game.WINDOW\_WIDTH - 100) *# TODO 7.修改敌机到达底部后返回顶部的策略，x随机生成，同初始化策略* self.y = random.randint(-Game.WINDOW\_HEIGHT, -68) *# TODO 7.修改敌机到达底部后返回顶部的策略，y随机生成，同初始化策略* """ TODO 8.原始敌机到达底部后返回顶部的策略删除  
 self.y = -68   
 """  
 else:  
 self.y += 4

class Game:  
 def model\_init(self):  
 self.background = Background(IMG\_BACKGROUND,0,0)   
 self.enemys = []   
 for \_ in range(5) :   
 self.enemys.append(EnemyPlane()) *# TODO 5.修改创建敌机操作* """ TODO 6.原始创建敌机操作被替换  
 self.enemys.append(EnemyPlane(IMG\_ENEMY,random.randint(0, Game.WINDOW\_WIDTH - 100 ),200))   
 """

#### 5.9敌机贴图随机选择制作分析

敌机创建和飞出屏幕时，敌机的图片需要随机生成，使用元组定义所有敌机的图片，敌机的贴图从元组中随机选择

#### 5.10代码实现5

APP\_ICON = "res/app.ico"

IMG\_BACKGROUND = "res/img\_bg\_level\_1.jpg"

*# TODO 1.设置敌机图片库常量元组*IMG\_ENEMYS = ("res/img-plane\_1.png","res/img-plane\_2.png","res/img-plane\_3.png","res/img-plane\_4.png","res/img-plane\_5.png","res/img-plane\_6.png","res/img-plane\_7.png")  
""" TODO 2.删除单独敌机图片  
IMG\_ENEMY = "res/img-plane\_5.png"

"""

class EnemyPlane(Model):  
 def \_\_init\_\_(self):  
 self.img = pygame.image.load(IMG\_ENEMYS[random.randint(0,6)]) *# TODO 3.设置图片路径随机从元组中获取* """ TODO 4.删除原始定义格式  
 self.img = pygame.image.load(IMG\_ENEMY) # 使用常量  
 """  
 self.x = random.randint(0,Game.WINDOW\_WIDTH - 100)   
 self.y = random.randint( -Game.WINDOW\_HEIGHT, -68)   
   
 def move(self):  
 if self.y > Game.WINDOW\_HEIGHT:  
 self.img = pygame.image.load(IMG\_ENEMYS[random.randint(0, 6)]) *# TODO 5.修改敌机到达底部后返回的图片随机生成，同初始化策略* self.x = random.randint(0, Game.WINDOW\_WIDTH - 100)  
 self.y = random.randint(-Game.WINDOW\_HEIGHT, -68)   
 else:   
 self.y += 4

#### 5.11实用技巧

1.Python程序在开发时，经常遇到使用的资源是限定范围内固定资源。遇到此情况，优先考虑使用元组将所有资源进行包装。在使用时，从预先定义好的元组中取数据即可。

2.如果资源在使用过程中会发生变化，优先考虑使用列表。

3.项目中的资源加载使用的是手工书写的形式，也可以使用推导式根据规律一次性加载。

IMG\_ENEMYS = ( "res/img-plane\_" + str(i) + ".png" for i in range(1,7))

### 玩家

#### 6.1玩家飞机制作分析

1.玩家的制作与敌机、背景的制作完全相同，区别之处是图片不同和贴图位置不同，模仿一架敌机或背景的制作完成玩家飞机的加入即可。飞机默认进入位置先暂定为200,500处

2.屏幕进行贴图时，按照先贴图的位置靠下，后贴图的位置考靠上的原则，因此贴图顺序要根据游戏规则合理设置

#### 6.2代码实现1

IMG\_PLAYER = "res/hero2.png" *# TODO 1.设置玩家飞机图片*

class Game:  
 def run(self):  
 self.frame\_init()   
 self.model\_init()   
 while True:  
 self.background.move()   
 self.background.display()   
 for enemy in self.enemys:   
 enemy.move()   
 enemy.display()   
 self.player.display() *# TODO 3.玩家飞机加入窗体中* pygame.display.update()   
 self.event\_init()   
  
 def model\_init(self):  
 self.background = Background(IMG\_BACKGROUND,0,0)   
 self.enemys = []   
 for \_ in range(5) :   
 self.enemys.append(EnemyPlane())   
 self.player = PlayerPlane(IMG\_PLAYER,200,500) *# TODO 2.初始化玩家飞机对象，放置在200,500*

#### 6.3玩家飞机移动控制制作分析

1.玩家飞机的移动由鼠标进行控制，因此获取到鼠标的位置即可进行玩家飞机的控制

2.获取鼠标的位置隶属于鼠标控制事件，对于界面操作有关的设置需要从pygame的文档中查询获取

3.鼠标位置与图片贴图位置之间存在着错位关系，需要进行横坐标与纵坐标的调整，在原始贴图位置基础上向左向下平移一定的距离，具体数值需要根据需求进行分析设计

4.相关文档

*# 12.鼠标移动事件*pygame.locals.MOUSEMOTION  
  
*# 13.获取鼠标位置*pygame.mouse.get\_pos()  
*# 返回值：元组，包含位置x与y坐标*

#### 6.4代码实现2

class Game:  
 def event\_init(self):  
 for event in pygame.event.get():   
 if event.type == pygame.locals.QUIT :   
 sys.exit()   
 if event.type == pygame.locals.MOUSEMOTION : *# TODO 1.设置监听鼠标移动事件* pos = pygame.mouse.get\_pos() *# TODO 2.获取鼠标位置坐标，结果是元组* self.player.x = pos[0] - 120/2 *# TODO 3.设置飞机中心位置在鼠标位置(x) 横坐标 - 飞机宽度1半* self.player.y = pos[1] - 78/2 + 5 *# TODO 3.设置飞机中心位置在鼠标位置(y) 纵坐标 - 飞机高度1半 +-微调数据*

### 子弹

#### 7.1发射子弹事件分析

1.发射子弹由鼠标按键控制，因此需要获取鼠标的按键状态，否则无法得知是否发射子弹，对于界面操作有关的设置需要从pygame的文档中查询获取

2.鼠标按键状态获取结果是一个元组，保存有三个数据，分别对应鼠标左中右三个按键的状态，因此需要先判断左键是否按下，才能进行子弹的发射动作

3.相关文档

# 14.获取鼠标按键按住状态  
pygame.mouse.get\_pressed()  
# 返回值：元组，包含左中右键是否被按下，0 - 未按下，1 - 按下

#### 7.2代码实现1

class Game:  
 def event\_init(self):  
 for event in pygame.event.get():   
 if event.type == pygame.locals.QUIT :   
 sys.exit()   
 if event.type == pygame.locals.MOUSEMOTION :   
 pos = pygame.mouse.get\_pos()   
 self.player.x = pos[0] - 120/2   
 self.player.y = pos[1] - 78/2 + 5   
 mouse\_state = pygame.mouse.get\_pressed() *# TODO 1.获取鼠标按键压下状态，返回得到元组，其中保存鼠标左中右键按下状态（1,0,0）* if mouse\_state[0] == 1 : *# TODO 2.判断左键是否按下* print("按下了鼠标左键")

#### 7.3子弹对象初始化分析

1.子弹在屏幕上不止一颗，因此在设计时，需要使用存储模型进行保存，推荐使用列表进行。

2.子弹可以制作独立的列表保存，如果后期制作多玩家飞机或敌人也发射子弹时，对于子弹的归属不便于区分，因此将子弹的列表归属到发射方是一种较好的设计原则，需要在玩家类中添加子弹的列表属性

3.当点击鼠标左键时，创建子弹，并加入列表

4.子弹对象的初始化位置需要由鼠标的位置确认，因此子弹的横坐标与纵坐标需要通过获取鼠标的位置计算得到

5.子弹的贴图操作在飞机贴图后进行，注意：传统设计原则是子弹归属飞机，在飞机的贴图操作中将子弹的贴图一并完成，本例采用单独贴子弹的方式完成。如果考虑到后期制作多玩家或敌机发射子弹的情况，建议将子弹贴图加入到归属对象的贴图操作中

#### 7.4代码实现2

IMG\_BULLET = "res/bullet\_13.png" *# TODO 1.设置子弹图片*

class PlayerPlane(Model):  
 *# TODO 2.覆盖init方法* def \_\_init\_\_(self,img\_path,x,y):  
 super().\_\_init\_\_(img\_path,x,y) *# TODO 3.调用父类init方法* self.bullets = [] *# TODO 4.定义子弹列表为空，默认没有子弹  
  
 # TODO 7.重写玩家飞机display方法* def display(self):  
 super().display() *# TODO 8.调用父类中对于飞机加入窗体的操作* for bullet in self.bullets: *# TODO 9.循环飞机中的子弹列表* bullet.display() *# TODO 10.调用子弹加入窗体的操作*

class Game:  
 def event\_init(self):  
 for event in pygame.event.get():   
 if event.type == pygame.locals.QUIT :   
 sys.exit()   
 if event.type == pygame.locals.MOUSEMOTION :   
 pos = pygame.mouse.get\_pos()   
 self.player.x = pos[0] - 120/2   
 self.player.y = pos[1] - 78/2 + 5   
 mouse\_state = pygame.mouse.get\_pressed()   
 if mouse\_state[0] == 1 :   
 pos = pygame.mouse.get\_pos() *# TODO 5.获取鼠标位置* self.player.bullets.append(Bullet(IMG\_BULLET,pos[0] - 20/2 ,pos[1] - 29 - 40)) *# TODO 6.创建子弹，加入飞机子弹列表，子弹横坐标为鼠标x坐标 - 子弹宽度1半，纵坐标为鼠标y坐标 - 子弹高度 +-微调数据*

#### 7.5子弹移动控制分析

1.子弹的移动操作通常设计速度较快，推荐设置为敌机速度的2到4倍

2.子弹一旦超出屏幕显示范围后将没有存在的价值，此时需要删除对应的子弹

3.子弹对象从列表中删除时，可能会引发删除失败的情况，推荐使用辅助列表保存要删除的子弹，创建独立的删除操作，二次删除对应的子弹

#### 7.6代码实现3

class PlayerPlane(Model):  
 def display(self):  
 super().display()   
 remove\_bullets = [] *# TODO 5.定义被删除的子弹列表* for bullet in self.bullets:   
 bullet.move() *# TODO 3.调用子弹的移动操作* bullet.display()   
 *# TODO 4.优化子弹存储队列，将超出屏幕的子弹删除* if self.y < -29: *# TODO 6.如果子弹位置超出屏幕* remove\_bullets.append(bullet) *# TODO 7.将要删除的子弹加入到待删除列表* for bullet in remove\_bullets: *# TODO 8.循环删除子弹列表* self.bullets.remove(bullet) *# TODO 9.从原始子弹列表中删除要删除的子弹*

class Bullet(Model):  
 *# TODO 1.定义子弹的移动方法* def move(self):  
 self.y -= 12 *# TODO 2.设定子弹的移动速度为12*

### 碰撞检测

#### 8.1碰撞检测分析

1. 子弹与敌机进行碰撞检测，需要书写在既有敌机列表也有子弹列表的位置，本例中将敌机列表传入玩家飞机的display操作，在display操作中做碰撞检测

2.子弹只有在屏幕内的时候才有可能碰撞到敌机，因此仅对未超出屏幕的敌机做检测

3.碰撞检测是检测两个矩形是否相交，对于界面操作有关的设置需要从pygame的文档中查询获取

4.碰撞检测只要获取到参与碰撞的两个对象对应的矩形即可进行

5.相关文档

# 15.创建矩形对象  
pygame.locals.Rect(x,y,w,h)  
# 参数1：整数，x坐标  
# 参数2：整数，y坐标  
# 参数3：整数，宽度  
# 参数4：整数，高度  
# 返回值：矩形对象

# 16.判断两个矩形是否相交  
pygame.Rect.colliderect(rect1, rect2)  
# 参数1：矩形对象，矩形1  
# 参数2：矩形对象，矩形2  
# 返回值：布尔值，矩形是否相交

#### 8.2代码实现1

class PlayerPlane(Model):  
 def display(self, enemys): *# TODO 2.在显示飞机和子弹的同时，传入敌机列表* super().display()  
 remove\_bullets = []  
 for bullet in self.bullets:  
 bullet.move()  
 bullet.display()  
 if self.y < -29:  
 remove\_bullets.append(bullet)  
 else: *# TODO 3.1.如果子弹位置未超出屏幕范围* bullet\_rect = pygame.locals.Rect(bullet.x,bullet.y, 20, 29) *# TODO 4.创建子弹矩形对象，传入x,y,width,height* for enemy in enemys: *# TODO 3.2使用子弹与所有敌机进行碰撞检测* enemy\_rect = pygame.locals.Rect(enemy.x, enemy.y, 100, 68) *# TODO 5.创建敌机矩形对象，传入x,y,width,height  
 # TODO 6. 调用碰撞检测方法，传入当前子弹对象，传入敌机对象，判断是否碰撞* if pygame.Rect.colliderect(bullet\_rect, enemy\_rect): *# TODO 7.如果碰撞* print("相撞了")  
  
 for bullet in remove\_bullets:  
 self.bullets.remove(bullet)

class Game:  
 def run(self):  
 self.frame\_init()   
 self.model\_init()   
 while True:   
 self.background.move()   
 self.background.display()   
 for enemy in self.enemys:   
 enemy.move()   
 enemy.display()   
 self.player.display(self.enemys) *# TODO 1.飞机加入窗体时，传入敌机列表实参self.enemys # 玩家飞机加入窗体中* pygame.display.update()   
 self.event\_init()

#### 8.3子弹与敌机碰撞处理分析

1.敌机被击中后敌机对象要做的处理很多，例如敌机位置的重置，图片重置等

2.子弹击中敌机后，子弹对象要进行删除

3.子弹一旦击中敌机，后续列表中的子弹无须对当前敌机进行再次判定

#### 8.4代码实现2

class PlayerPlane(Model):  
 def \_\_init\_\_(self,img\_path,x,y):  
 super().\_\_init\_\_(img\_path,x,y)   
 self.bullets = []   
 def display(self, enemys):   
 super().display()   
 remove\_bullets = []  
 for bullet in self.bullets:   
 bullet.move()   
 bullet.display()   
 if self.y < -29:   
 remove\_bullets.append(bullet)   
 else:   
 bullet\_rect = pygame.locals.Rect(bullet.x,bullet.y, 20, 29)   
 for enemy in enemys:   
 enemy\_rect = pygame.locals.Rect(enemy.x, enemy.y, 100, 68)   
 if pygame.Rect.colliderect(bullet\_rect, enemy\_rect):   
 enemy.img = pygame.image.load(IMG\_ENEMYS[random.randint(0, 6)]) *# TODO 1.敌机被击中后恢复初始状态* enemy.x = random.randint(0, Game.WINDOW\_WIDTH - 100) *# TODO 1.敌机被击中后恢复初始状态* enemy.y = random.randint(-Game.WINDOW\_HEIGHT, -68) *# TODO 1.敌机被击中后恢复初始状态* remove\_bullets.append(bullet) *# TODO 2.将产生碰撞的子弹加入删除列表* break *# TODO 3.当前子弹击中了一架敌机，终止对剩余敌机的碰撞检测，终止敌机循环* for bullet in remove\_bullets:   
 self.bullets.remove(bullet)