

# Python语言与系统设计报告

设计题目：飞机大战小游戏

学生姓名：叶春亭

专 业：计算机科学与技术

班 级：计科20-01班

学 号：2020215471

指导教师：李培培

完成日期：2021年12月23日

# （一）需求和规格说明

**问题描述：**飞机大作战相信我们小时候都玩过，该程序设计了一个简易的飞机大战小游戏

**编程任务：**

1. 做出一个飞机大作战的界面，该程序用的时pygame模块
2. 该小游戏包含我方英雄飞机，可以通过键盘上下左右移动，子弹可以为自动发射模式
3. 还需要敌机的参与，敌机随机出现在游戏界面中，并且随机发射子弹，当我方英雄子弹打中敌机后，敌机摧毁，同样，我方英雄被敌机子弹射中时，该局游戏即结束
4. 其他的内容可以自行发挥

# （二）设计

## 1．设计思想

（1）首先我们这里利用的是pygame模块来实现游戏界面的搭建。

（2）对于一个小游戏的图像加载、图像的位置变换、绘制图像都需要我们编写代码来一个个地处理，但是如果我们的游戏需要处理很多张图像，那么代码内部将会非常复杂 ，因为我们要在每一个游戏循环中，一个图像移动一次就需要写一次代码。那么需要如何解决这个问题呢？ pygeme中有一个很重要的概念，就是Sprite（精灵）以及精灵组（Sprites Group）。pygame.sprite.Sprit类

即为精灵类，这是一个抽象的类，也就是说，必须通过派生该类的子类，才能使用，我们先看一些该类的属性方法：



考虑到我们的飞机大作战需要很多移动的物体：很多架敌机、敌机发射的子弹也很多、以及我方英雄飞机、我方发射的很多子弹，并且游戏中每一个物体也就是游戏的构成（我们应该下意识的把它们当作是一个个精灵）的运动方式都是不同的，比如，英雄可以上下左右移动，而子弹只能上下移动，敌机只能左右移动，Sprite类就很好的解决了我们的问题，从图中我们可以看到，Sprite类有两个属性，image用来储存图片，也就是该精灵的图片，rect（矩形）则用来保存该图片的位置以及大小（即矩形的尺寸）；还有一个很重要的方法update(),继承精灵类的每一个子类可以通过重写该方法，来达到更新动作不同的目的，举个例子：我方英雄飞机通过检测按下了键盘左键的事件来让飞机调用重写之后的update方法，这样就可以使飞机向左移动相应的位置，这个移动位置的操作就是在update方法中实现的，而kill()方法就是“杀死”该精灵，让该精灵消失，可以在后面的飞机销毁中用到。

（3）还有一个概念，就是精灵组：



当有很多同类精灵时，可以创建一个精灵组，添加进精灵后，来同时调用该精灵组中的update()方法以及draw()方法，draw方法主要是将该精灵“画”在屏幕(screen)上。

（4）最后在一个循环中循环的调用这些精灵的update()来更新位置，以及一些事件的监测，最终再加上一些程序的优化以及功能的增添。

## 2. 设计表示

1. 项目结构：
2. 为了便于代码方便调试及后期维护，以及使项目结构更加清晰，因此此将该项目分为PlaneWar\_Main.py和PlaneWar\_Sprites.py，以及一个images的文件夹三部分。
3. 在Sprites.py文件中，主要是通过继承pygame.sprite.Sprite类来实现了敌我飞机和子弹，以及其他的一些优化的精灵类,并且重写了所有实际精灵类的update()方法,以便在Main文件中使用。
4. 而PlaneWar\_Main.py文件中主要只有一个Game类,用来初始化pygame模块以及界面的搭建,还有创建精灵以及一些事件的监听和实现,还包含了游戏中的主要循环函数,最后还有main程序中的一些功能的优化代码。
5. 最后就是images文件夹中包含了所有飞机以及子弹的图片，还有背景，以及飞机被摧毁后的一系列爆炸图，最后还有一首背景音乐MP3文件。

1. PlaneWar\_Sprites.py 文件中的相关类与结构说明：

它们的类图分别如下所示：

GameSprite类：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **类名** | **成员类别** | **成员名** | **返回类型/默认类型** | **描述** |
| GameSprite | 函数 | \_\_init\_\_ | None | 初始化函数 |
| update() | None | 更新函数  （抽象函数） |
| 数据 | image | Str | 保存精灵的图片 |
| rect | Tuple | 保存精灵的位置 |

Background类

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **类名** | **成员类别** | **成员名** | **返回类型/默认类型** | **描述** |
| Background | 函数 | \_\_init\_\_ | None | 初始化函数 |
| update() | None | 更新函数  （抽象函数） |
| 数据 | image | Str | 保存精灵的图片 |
| rect | Tuple | 保存精灵的位置 |

EnemyBullet类

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **类名** | **成员类别** | **成员名** | **返回类型/默认类型** | **描述** |
| EnemyBullet | 函数 | \_\_init\_\_() | None | 初始化函数 |
| update() | None | 更新函数  （抽象函数） |
| 数据 | image | Str | 保存精灵的图片 |
| rect | Tuple | 保存精灵的位置 |

EnemyPlane类

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **类名** | **成员类别** | **成员名** | **返回类型/默认类型** | **描述** |
| EnemyPlane | 函数 | \_\_init\_\_() | None | 初始化函数 |
| L\_Or\_R() | Str | 用于产生1和0 随机数来决定该敌机是从左侧飞出还是从右侧飞出 |
| moveLeft() | None | 敌机向左移动 |
| moveRight() | None | 敌机向右移动 |
| Update() | None | 更新敌机的位置 |
| 数据 | speed | Int | 确定敌机移动速度 |
| LOR | Str | 确定敌机出现的位置,左或者又 |
| shootX | Int | 随机一个位置,即敌机出现位置的纵坐标 |
| isAlive | Bool | 判断敌机是否阵亡 |
| count | Int | 用作爆炸时飞机图片更新计数 |

Hero类

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **类名** | **成员类别** | **成员名** | **返回类型/默认类型** | **描述** |
| Hero | 函数 | \_\_init\_\_() | None | 初始化函数 |
| moveLeft | None | 英雄向左移动 |
| moveRight | None | 英雄向又移动 |
| moveUp | None | 英雄向上移动 |
| moveDown | None | 英雄向下移动 |
| Shoot | None | 英雄发射子弹函数 |
| Update() | None | 英雄飞机更新位置 |
| 数据 | speed | Int | 设置英雄飞机移动速度 |
| bullets | Group | 英雄发射子弹的精灵组 |
| isAlive | Bool | 判断英雄是否存活 |
| count | Int | 用作爆炸时飞机图片更新计数 |
| Image | Image | 用于保存飞机图片 |

Button类

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **类名** | **成员类别** | **成员名** | **返回类型/默认类型** | **描述** |
| Button | 函数 | \_\_init\_\_() | None | 初始化函数 |
| setPosition() | None | 根据按钮类型设置位置 |
| update() | None | 更新函数 |
| 数据 | image | Str | 保存精灵的图片路径 |
| rect | Tuple | 保存精灵的位置 |

PlaneWar\_Main.py文件中相关代码结构与类图表如下:

PlaneGame类

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **类名** | **成员类别** | **成员名** | **返回类型/默认类型** | **描述** |
| PlaneGame | 函数 | \_\_init() | None | 初始化函数 |
| createSprotes() | None | 创建精灵/组函数 |
| startGame | None | 游戏开始,游戏中的主要循环代码封装在这个函数中 |
| showScore() | Node | 展示游戏中获得的积分 |
| eventHandle() | None | 键盘事件监听的函数,用来控制英雄移动以及创建敌机等操作 |
| enemyShoot | None | 用于敌机发射子弹 |
| Update() | None | 用于所有精灵组的update和draw操作 |
| isMouseIn(button) | Bool | 用于判断鼠标是否停留在精灵上,用于后续的mouse事件检测 |
| checkCollide() | None | 用于飞机与子弹间的碰撞检测,即飞机撞毁的检测 |
|  | 数据 | screen | Surface | pygame的屏幕对象 |
| clock | Clock | 时钟对象,用于敌机的生成 |
| score | Int | 用于记录游戏中所得的分数 |
| Background1、2 | Background | 游戏背景的对象 |
| Back\_ground | Group | 游戏背景的精灵组 |
| Enemy\_group | Group | 存放敌机的精灵组 |
| hero | Hero | 存放英雄的精灵 |
| font | Font | 创建的字体对象 |
| collideDict | Dict | 用于存放敌机与英雄子弹之间的碰撞之后的精灵对象的字典 |
| collideBullets | List | 用于存放英雄与敌机子弹之间的碰撞的信号 |
| enemyBullets\_group | Group | 存放敌机子弹的精灵组 |

## 3. 游戏中各模块的实现以及优化

1. 首先是各个精灵类的基本实现，每个精灵类都直接或间接继承了pygame.sprite.Sprite类，\_\_init\_\_函数中都有一个imageName参数来作为该精灵的图片路径，而函数内部则运用pygame.image.load()函数来载入该路径的图片,作为该精灵的图片,而self.rect则是该图片的矩形尺寸(包含坐标和宽高),而update函数的重写则是该部分的重点,每类精灵根据需求的不同而重写不同的update方法,下面我详细介绍一下:
2. 敌机类(EnemyPlane),敌机我们要求随机从窗口上方的左侧或者右侧以一定的speed向另一侧移动,因此我们首先得确定两个数据,一是左或者又(LOR),二是出现位置的纵坐标(self.rect.y),因此我们用了两次random.randint()函数来随机出这两个数据,update函数中就通过判断起始来向左或者向右移动,还有一个点:我们需要确定每个敌机发射子弹的位置,以便后续子弹的发射,这里同样运用随机数来确定shootX。

代码优化:当敌机被我方子弹击中时,我们想要有一个动态爆炸的效果,通过考究,我们通过不断更新敌机的image来快速“替换”来达到动态的效果，因此我们设置一个count用来计数，每一次循环都更新一次图片，这样当帧率够高时，呈现的就是一个动态效果，同样英雄的爆炸也是如此。

1. 子弹类(Bullet),不管是敌机子弹还是英雄发射的子弹,update()中都是确定子弹发射的初始位置以及位置的更新,这里以英雄子弹为例,就不过多赘述:

class HeroBullet(Bullet):

    '''英雄子弹类'''

    def \_\_init\_\_(self,hero) -> None:

        self.imageName = './PlaneWar/images/bullet2.png'

        super().\_\_init\_\_(self.imageName)

        self.rect.x = hero.rect.x + 48.5

        self.rect.y = hero.rect.y - 11

        self.speed = 5

        pass

    def update(self):

        self.rect.y -= self.speed

        if self.rect.y <= 0:

            self.kill()

            pass

        pass

1. 英雄飞机类(Hero),英雄的位置变换是根据用户键盘的press来实现,因此这一部分主要是利用事件的监听,而该类中的update 函数首先用了上述的更新图片路径的方法实现动态的爆炸效果,其次就是有四个函数分别控制英雄的上下左右移动,而调用这些函数的地方就是键盘事件发生的时候。
2. 在PlaneGame类中，封装了游戏的主要进程，我首先介绍一下背景音乐的加入:

pygame.mixer.init() #初始化

        pygame.mixer.music.load('./PlaneWar/images/background.mp3')  #加载音乐路径

        pygame.mixer.music.set\_volume(0.2)

        pygame.mixer.music.play(-1) #音乐循环次数

首先调用pygame.mixer.init()函数进行mixer模块的初始化,然后通过load()函数并传入背景音乐文件的路径,加载音乐,之后用set\_volume()设置音量,最后play函数设置音乐循环次数,-1代表无限循环。

3)对于游戏计分的实现: 

我们利用了系统的文字类SysFont来实现:

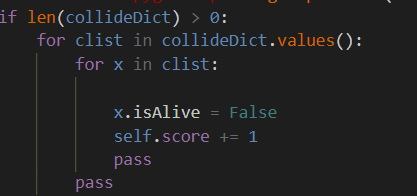
def showScore(self):

        txt = str(self.score)

        font = pygame.font.SysFont("freesansbold.ttf", 30)  # 30:font size

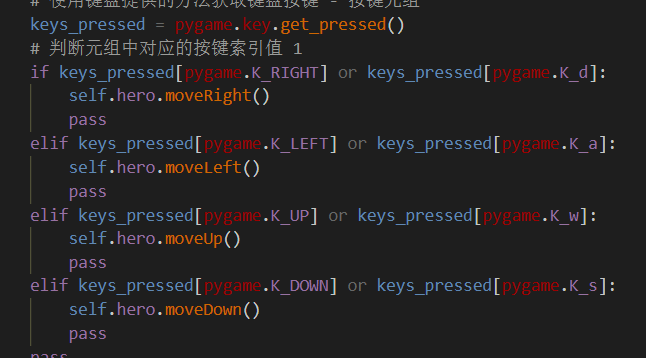
        text = font.render(txt, True, (0,0,0))  # (0,0,0) color of font

        self.screen.blit(text,(10,10))  # (10,10) rect left top

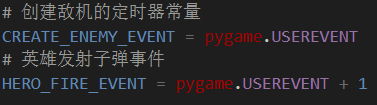
首先在PlaneGame类中设置了一个属性score = 0,用来记录这局游戏的分数, 

在后续敌机与我方英雄子弹的碰撞检测中,每击毁一架敌机,分数就加一分,然后设置好字体大小等数据后将text文本blit()到self.screen上,并将此函数加入游戏主循环,以达到时刻改变分数的目的。

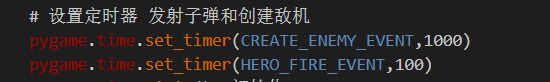
4）对于键盘事件的监听，也就是玩家操控英雄移动的模块，利用的是键盘(key)模块提供的方法get\_pressed()来返回一个元组,这个元组包含所有按键的值,如果某按键索引值为1,则代表该按键被pressed,因此一次来完成英雄飞机的移动,代码如下:



5)另外还有两个功能需要用到事件监听,创建敌机和英雄自动发射子弹,由于这两个功能都是以时间作为信号,不是我们认为操控的,因此我们设置两个自定义事件类型

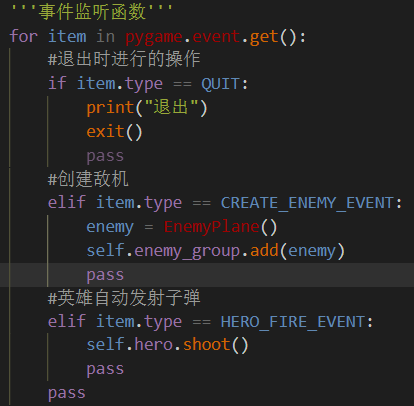


并在该游戏类的初始化函数中为这两个事件设置两个定时器:



传入的第二个参数表示每隔这么多毫米秒就触发一次该事件,回到事件检测中,我们利用for item in pygame.event.get()循环,在该循环中get()返回的是一个事件列表,这个列表会实时更新最先发生的事件,如果事件的类型属于

这两个事件,则执行相应的操作,代码如下:



6)碰撞检测问题(Collide),也是本项目的重点和难点,首先我们应该了解两个pygame模块提供的碰撞检测的函数:

① pygame.sprite.groupcollide(group1,group2,dokill,dokill)

这个函数用于两个精灵组之间的碰撞检测,前两个参数group1,group2为需要检测的两个精灵组(Group),而后面两个参数则是两个bool类型的,需要传入True或者False来决定两精灵碰撞后是否对碰撞的精灵调用kill()方法,而这个函数的返回值是一个Dict(字典),字典的keys是group1中所有与group2中精灵碰撞的sprite对象,而每个key对应的value值则是在group2中与该key(sprite对象)发生碰撞的所有sprites,用一个list来封装这些sprite,所以我们用这个函数还判断敌机与英雄子弹之间的碰撞,因为所有英雄子弹和敌机都封装在一个Group中。代码如下:

collideDict = pygame.sprite.groupcollide(self.enemy\_group,self.hero.bullets,False,True)

        if len(collideDict) > 0:

            for enemy in collideDict:

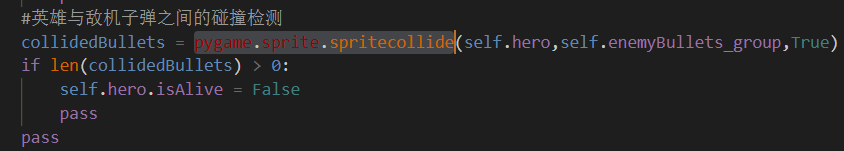
                enemy.isAlive = False

                self.score += 1

                pass

功能优化：由于我们之前对于飞机爆炸动态的功能优化中,进行的操作是将被击中的精灵中image的图片路径”替换”来达到效果的,但是碰撞检测是很快的,当飞机被炸毁时,第一个子弹被kill,但此时,我们并不能将敌机kill,因为我们想要一个动态的效果,需要后续再将这个精灵更改图片呈现出来,但这样就会出现一个问题:如果英雄子弹比较密集,在敌机呈现爆炸动图时其他子弹打中这个爆炸的飞机,仍然会触发碰撞检测,这样的话self.score就会重复增加,针对这个问题,我的解决方法是:将爆炸后的精灵的rect.size()调整为(0,0),也就是将这个精灵变消失,但是图片没有消失,只是一个精灵的虚拟大小变成0了,所以哪怕后续子弹再撞到炸毁飞机的图片上,也不会触发碰撞检测,因为碰撞检测针对的是精灵,也就是sprite的rect,而不是实际的图片。

②pygame.sprite.spritecollide(Sprite,Group,dokill)这个函数用于一个Sprite和一个Group之间的碰撞检测,当Sprite和Group中的精灵碰撞后,会返回一个list,储存Group中所有与之碰撞的精灵,并且按照dokill参数对这些精灵调用kill()方法,这个方法用来判断英雄与敌机子弹之间的碰撞检测,不多赘述,具体代码如下:

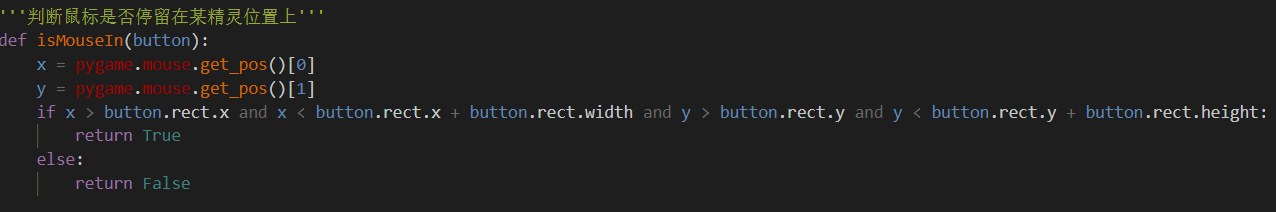


7)最后为了让游戏功能更完善,还增加了两个按钮,也就是一局游戏结束后,弹出的界面,如下图:



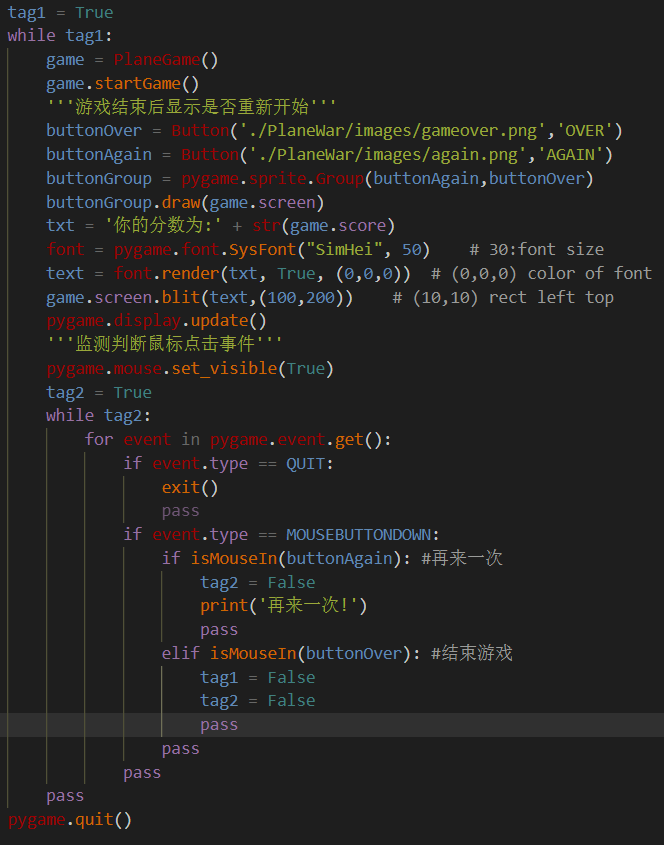
重新开始和结束游戏按钮,对于按钮功能的实现,有很多方法,这里讲一下我的方法:首先创建一个Button类,这个类也继承Sprite类作为一个精灵,传入按钮的image,并根据按钮类型的不同确定位置。然后再用一个mouse事件的检测来判断是否按下某个按钮,具体分为一下步骤:

* + - 1. 首先写一个判断鼠标是否停留再某精灵位置上的函数,



该函数通过pygame模块的pygame.mouse.get\_pos()方法来返回鼠标悬停的位置坐标,进一步判断,并返回bool值。

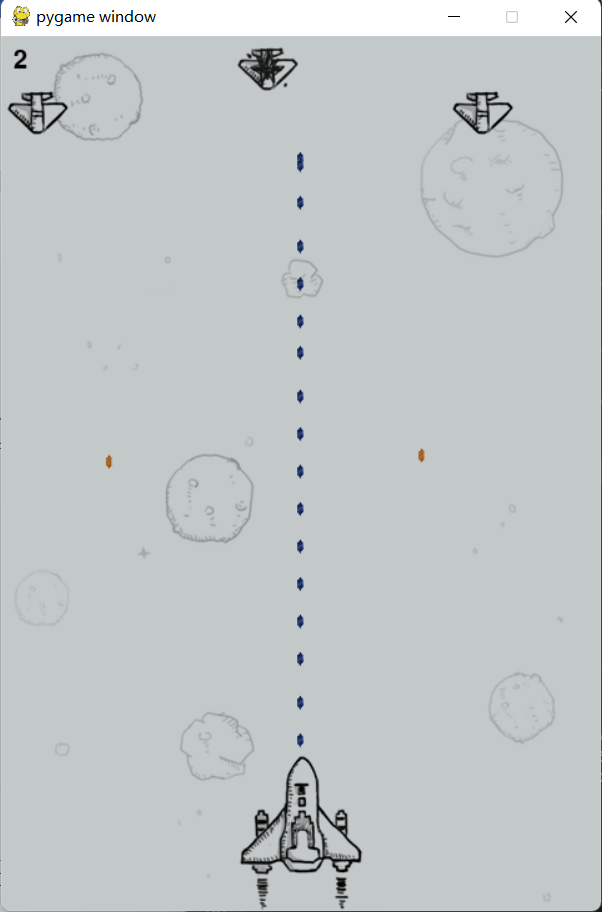
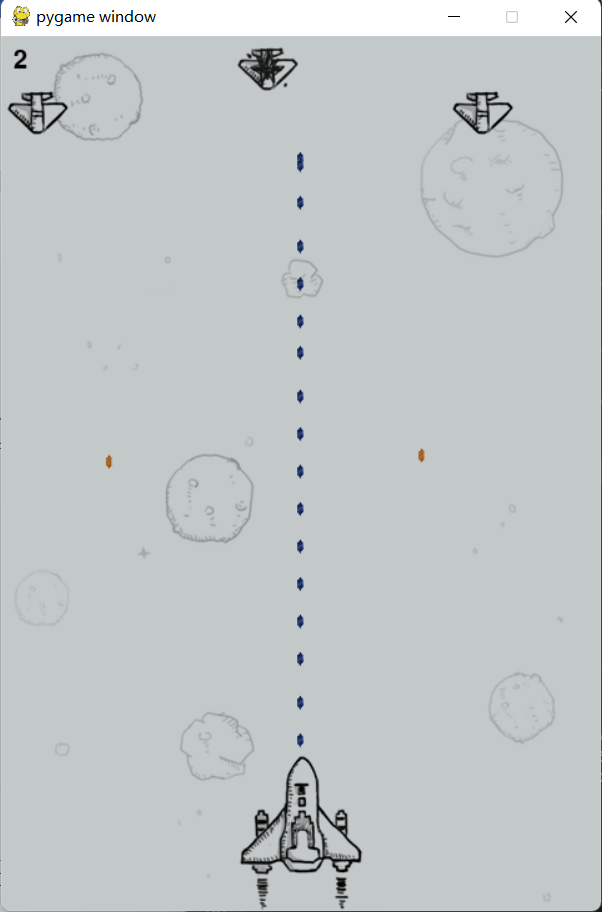
②之后在\_\_main\_\_中利用pygame.event.get()来判断MOUSEBUTTONDOWN(鼠标按下)类型的事件的监测,如果用户单击的再来一次,则通过tag2 = False跳出鼠标事件检测循环,并执行\_\_main\_\_中重新开始的循环,而若点击结束游戏,则将两个循环都跳出,结束程序。具体代码如下：



其中的最终分数也是利用blit()函数，和上面的score显示方法一致，不多赘述。

# （三）用户手册

（1）运行前肯定需要下载pygame模块，将DOS窗口打开，输入pip install pygame，安装成功后可以运行PlaneWar\_Main.py文件，运行之后出现如下界面：



之后可以通过键盘的WASD或者小键盘的上下左右来移动下方的英雄飞机，移动飞机将敌机击落的同时注意规避敌机发射的子弹。

1. 如果不小心被敌机击中，则会弹出如下画面：

此时若点击重新开始按钮，则会重新开始，若点击结束游戏，则程序运行结束。

# （四）调试及测试

该游戏运行调试的视频我一同打包在Vedio文件中，请老师查看。

# (五) 感想

通过本次实验，首先让我见识到了python的强大之处，它包含丰富的拓展库和接口，使得python可以用来做许多事，比如本项目用到的pygame，可以很方便的用来做小游戏，当然，最重要的也是让我学会了利用pygame来写小游戏，首先我认识到，任何动态视频或动画，包括小游戏，都是通过许多图片以一定的帧率呈现在我们眼前，这样就形成了动态感，当一个游戏中有许多物体需要这种动态感时，pygame提供了一个几乎完美的类Sprite，来方便我们管理和运用这些物体，同样，学习过程中也遇到过许多困难，比如物体的碰撞检测，还有游戏循环主框架的构建，以及其他一些算法方面的问题，当然也让我对面向对象编程有了更进一步的提升，同时也让我的学习能力变得更强，总之这个大作业对我的帮助还是很大的。