# 这一节我们来学习碰撞检测

# 1.碰撞检测的讲解过程说明p78

|  |
| --- |
|  |

# 2.sprite的碰撞检测方法p79

|  |
| --- |
|  |
|  |

# 2.2.碰撞检测的基本实现

## 思路

|  |
| --- |
|  |
|  |

## 代码实现，只是来用学习基本原理，游戏项目并不适用

## 1.在Game类的初始化方法的最后添加如下代码，锚点是适当敌机不会动，然后用英雄飞机去撞击敌机

|  |
| --- |
|  |

## 2.给Game类实现一个检测碰撞的方法，代码如下：

|  |
| --- |
|  |

## 3.在游戏循环里面调用游戏碰撞检测方法，注意代码一定要放在精灵组更新的代码之前，否则无效

|  |
| --- |
|  |

## 测试一下

|  |  |
| --- | --- |
|  |  |

## 爆炸是爆炸了，但是英雄飞机还没有碰到敌机，敌机就先引爆了，这个感觉不太好

## 为什么还有碰撞到，敌机就会爆炸？

## 因为其实飞机精灵有一个外接矩形的，敌机也是一样，上面的方法是用精灵的矩形来判断是否发生碰撞，所以这种方法其实不太好

|  |
| --- |
|  |
|  |

# 3.实现高质量的碰撞检测p80

## 从上面的测试例子可以看到，使用矩形的碰撞检测有时候并不靠谱，我们需要使用更加好的方法。

## 1.方法说明

|  |
| --- |
|  |
|  |
|  |
|  |

## 2.代码实现：回到game.py模块，修改碰撞检测方法的代码，把最后一个参数改为pygame.sprite.collide\_mask

|  |
| --- |
|  |

### 效果：这样子修改后效果好很多

## 3.为了提供执行效率，在创建精灵时我们需要给精灵添加一个mask属性，由于我们使用的精灵都是继承自GameSprite的。我们只需要在这个父类的构造方法里面添加mask就行了，因为子类在创建对象的时候会先调用父类的构造方法，我们在GameSprite的初始化方法里面添加这个mask属性

|  |
| --- |
|  |

## 效果对比

|  |  |
| --- | --- |
| **没有遮罩** | **有遮罩** |
|  |  |
|  |  |

### 可见，有了mask碰撞检测就很准确高效了

# 4.英雄飞机被碰撞的注意点介绍p81

## 在上面的碰撞检测中，我们只对敌机做了爆炸处理，其实当英雄飞机撞击到敌机，英雄飞机也会爆炸，我们啦学习这个功能，

## 1.先理清一下思路

|  |
| --- |
|  |
|  |
|  |

# 5.玩家飞机撞毁的处理p82

## 1.我们把让敌机静止的代码删除，让玩家飞机的速度恢复为5

|  |
| --- |
|  |

## 2.修改Game类的check\_collide方法，在发生碰撞后不仅仅敌机会被摧毁，英雄飞机也会被摧毁

|  |
| --- |
|  |

# 6.正在爆炸的飞机不会影响英雄飞机p83

### 上面存在问题

|  |
| --- |
|  |
|  |

## 1.为了保证英雄飞机不会被敌机爆炸的残骸摧毁，我们回到Game类，为碰撞检测方法添加一个过滤残骸的代码

|  |
| --- |
|  |

## 2.效果：保证敌机残骸不会摧毁英雄飞机

# 7.自定义事件的需求说明p84

## 说明，我们上面在英雄飞机牺牲后直接调用面板类更新画面是笔记麻烦的，我们需要使用自定义事件

|  |
| --- |
|  |

# 8.英雄飞机被撞击后控制面板的更新p85

## 1.在game\_items.py中定义一个英雄牺牲的事件的全局常量

|  |
| --- |
|  |

## 2.在Hero类中重写父类的reset\_plane方法，重置飞机后需要发布英雄牺牲事件，通知主程序更新界面

|  |
| --- |
|  |

## 3.在game.py模块的handler事件中添加对自定义事件的处理代码

|  |
| --- |
|  |

## 效果

|  |
| --- |
|  |

# 9.英雄飞机爆炸后的无敌时间和位置重置p86

## 说明

|  |
| --- |
|  |

## 代码实现

## 1.在game\_items.py模块里面定义一个英雄无敌时间结束事件

|  |
| --- |
|  |

## 2.在Hero类的reset\_plane方法下面对这个事件发送定时器，到时时间是3秒也就是3000毫秒

|  |
| --- |
|  |

## 3.需要在Game类在添加对这个事件的响应代码，注意在此之前，必须在reset\_plane方法一开头把is\_power设置为True

|  |
| --- |
|  |

## 效果，英雄飞机被摧毁后新登场的英雄飞机有3秒无敌时间

## 4.在Game类的reset\_game方法里面需要重置英雄飞机的位置

|  |
| --- |
|  |

### 效果：当游戏结束后即使英雄飞机在右边死的，按空格都可以把它定位到游戏开始位置

# 10.子弹类的功能说明p87

|  |
| --- |
|  |

# 11.定义子弹类p88

## 思路

|  |
| --- |
|  |
|  |

## 代码实现

## 1.在game\_items.py模块中定义一个子弹类继承GameSprite，需要重写父类的update方法

|  |
| --- |
|  |

# 12.发射子弹的定时事件处理p89

## 思路

|  |
| --- |
|  |
|  |

## 代码实现

## 1.在game\_items.py模块里面定义一个英雄飞机发射子弹的自定义事件

|  |
| --- |
|  |

## 2.在Hero类的初始化方法最后添加用定时器发送发射子弹的事件的代码

|  |
| --- |
|  |

## 3.在game.py模块的event\_handler方法里面添加对英雄发射子弹事件的响应代码，我们先添加一句打印输出，测试一下

|  |
| --- |
|  |

### 效果：只要游戏没有结束，就会每隔 0.2秒输出一句话

|  |
| --- |
|  |

## 4、给英雄飞机添加一个发射子弹的方法

|  |
| --- |
|  |

## 5. 在game.py模块的event\_handler方法里面修改对英雄发射子弹事件的响应代码

|  |
| --- |
|  |

### 效果：只要游戏没有结束，就会每隔 0.2秒输出一句话

|  |
| --- |
|  |

# 13.飞机发射子弹的代码实现p90

## 思路

|  |
| --- |
|  |

## 代码实现

## 1.把上面Hero类的fire方法代码修改如下

|  |
| --- |
|  |

## 2.测试一下

|  |
| --- |
|  |

# 14.子弹击中敌机的碰撞检测p91

## 思路

|  |  |
| --- | --- |
|  |  |
|  |  |

# 15.子弹击中敌机的代码实现p92

## 代码实现：

## 1.回到game.py模块，找到碰撞检测的代码，这个方法里面添加子弹打中敌机的碰撞检测代码

|  |
| --- |
|  |

## 2.测试一下，效果如图

|  |
| --- |
|  |

# 16.游戏重置的完善 p93

## 打开game.py模块，找到reset\_game方法，把它完善一下

|  |
| --- |
|  |

## 效果

|  |
| --- |
|  |
|  |

# 17.弹药的处理方式说明p94

## 思路

|  |
| --- |
|  |
|  |
|  |
|  |
|  |

# 18.定义道具类p95

## 代码实现

## 1.在game\_items.py中定义一个道具类，继承自GameSprite

|  |
| --- |
|  |

# 19.抛出道具p96

|  |
| --- |
|  |
|  |

## 代码实现

## 1.在game\_items.py文件在定义一个抛出道具事件

|  |
| --- |
|  |

## 2.在game.py模块中给Game类添加一个create\_supplies方法

|  |
| --- |
|  |

## 3.在Game类的初始化方法最后定义创建道具方法

|  |
| --- |
|  |

## 4.在Game类的event\_handler方法中添加对头发道具事件的处理代码

|  |
| --- |
|  |

## 测试一下，

|  |
| --- |
|  |

## 是会出现道具的，说明这个方法没有问题

# 20.英雄飞机拾取道具p97

## 思路

|  |
| --- |
|  |
|  |
|  |

## 代码实现

## 1.在game\_items.py模块里面添加子弹增强失效事件

|  |
| --- |
|  |

## 2.在game.py模块的Game类的碰撞检测方法的后面添加英雄拾取道具的代码

|  |
| --- |
|  |

## 3.在Game类的event\_handler方法中添加对双排子弹失效事件的处理代码

|  |
| --- |
|  |

## 效果：捡到子弹增强会有20秒的双排子弹，过后又是单排子弹，拾取到炸弹炸弹数量会增加

# 这一节学习到此为止，这一节的各个模块的完整代码如下

## game.py

|  |
| --- |
| #游戏主程序  import pygame  import random  from game\_items import \*  from game\_hud import \*  from game\_music import \*  class Game(object):      def \_\_init\_\_(self) -> None:         #1.游戏主窗口         self.main\_window = pygame.display.set\_mode(SCREEN\_RECT.size)         #设置窗口标题         pygame.display.set\_caption('飞机大战')         #2.游戏状态属性         self.is\_game\_over = False #游戏结束标记         self.is\_pause = False    #游戏暂停标记         #3.创建精灵组，3个精灵组：all\_group,enemies\_group,supplies\_group         self.all\_group = pygame.sprite.Group() # 存放所有精灵的精灵组         self.enemies\_group = pygame.sprite.Group() # 敌人精灵组         self.supplies\_group = pygame.sprite.Group() # 补给精灵组或者道具精灵组           #4.创建精灵         #创建背景精灵，需要2个精灵否则无法实现连续滚动效果，默认向下移动,我们将他们添加到all\_group中         #写法1      #    BackgroundSprite(False,self.all\_group) #默认在窗口里面显示      #    BackgroundSprite(True,self.all\_group)  #默认在窗口顶部外面显示         #写法2         self.all\_group.add( BackgroundSprite(False),BackgroundSprite(True))           #创建游戏控制面板         self.hud\_panel = HUDPanel(self.all\_group)         #创建英雄精灵,有速度的         self.hero = Hero(self.all\_group)         #在创建英雄飞机后修改炸弹数量、         self.hud\_panel.show\_bomb(self.hero.bomb\_count)           # 初始化敌机         self.creat\_enemies()         # 创建道具         self.creat\_supplies()         #5.音乐播放器        def reset\_game(self):        """重置游戏"""        self.is\_game\_over = False #游戏结束标记        self.is\_pause = False    #游戏暂停标记        self.hud\_panel.reset\_panel() #重置显示面板        # 重置英雄飞机的位置        self.hero.rect.midbottom = HERO\_DEFAULT\_MID\_BUTTOM        #销毁所有的敌机        for enemy in self.enemies\_group.sprites():            enemy.kill()        # 销毁所有子弹        for bullet in self.hero.bullets\_group.sprites():            bullet.kill()        # 重新创建敌机        self.creat\_enemies()        def creat\_enemies(self):          """创建敌机"""          # 先获取敌人精灵组的精灵数量          count = len(self.enemies\_group.sprites())          # 再定义精灵需要添加到的精灵组，有all\_group和enemies\_group          groups = (self.all\_group,self.enemies\_group)          #根据游戏级别和敌机数量来创建敌机          if self.hud\_panel.level ==1 and count == 0: # 关卡1              for i in range(16):                  Enemy(0,3,\*groups)          elif self.hud\_panel.level ==2 and count == 16: # 关卡2              # 1.先给敌机加速              for enemy in self.enemies\_group.sprites():                  enemy.max\_speed = 5  # 此时只有一种敌机              # 2.创建敌机              for i in range(8):                  Enemy(0,5,\*groups) # 创建8架小飞机              for i in range(2):                  Enemy(1,1,\*groups) # 创建2架中飞机          elif self.hud\_panel.level ==3 and count == 26: # 关卡3              # 1.先给敌机加速              for enemy in self.enemies\_group.sprites():                  enemy.max\_speed = 7 if enemy.kind == 0 else 3 #此时有两种敌机              # 2.创建敌机              for i in range(8):                  Enemy(0,5,\*groups) # 创建8架小飞机              for i in range(2):                  Enemy(1,1,\*groups) # 创建2架中飞机              for i in range(2):                  Enemy(2,1,\*groups) # 创建2架大飞机      def creat\_supplies(self):          """创建道具"""          Supply(0,self.supplies\_group,self.all\_group)          Supply(1,self.supplies\_group,self.all\_group)          # 每隔30秒发送一次抛出道具事件          # pygame.time.set\_timer(THROW\_SUPPLY\_EVENT,5000) # 这里是为了测试方便把时间间隔暂时改为10秒          pygame.time.set\_timer(THROW\_SUPPLY\_EVENT,30000)      def start(self):          """开始游戏"""          #创建游戏时钟          clock = pygame.time.Clock()          # 定义应该逐帧动画计算器，用来控制逐帧动画的播放速度          frame\_counter = 0          #游戏主循环          while True:              # 游戏循环的第一件事就是判断游戏是否结束，当玩家的飞机数量是0，就可以              self.is\_game\_over = self.hud\_panel.lives\_count == 0              #监听事件，调用event\_handler方法              if self.event\_handler(): #返回True说明用户点击了关闭按钮                  self.hud\_panel.save\_best\_score() #游戏介绍之前需要把最好成绩保存到record.txt文件中                  return              # 根据游戏状态切换界面显示的内容              if self.is\_game\_over:                  self.hud\_panel.panel\_pause(True,self.all\_group) # Game over了，游戏需要停止注意第一个参数和暂停不一样              elif self.is\_pause:                  self.hud\_panel.panel\_pause(False,self.all\_group) #如果暂停标志位真，就暂停游戏              else:                  self.hud\_panel.panel\_resume(self.all\_group) # 从暂停状态恢复                  # 获取当前时刻的按键元组                  keys = pygame.key.get\_pressed()                    # 我们利用一个比较简便的方法来判断水平移动的方向，                  # 就是用keys[pygame.K\_RIGHT]-keys[pygame.K\_LEFT]                  move\_hor = keys[pygame.K\_RIGHT]-keys[pygame.K\_LEFT]                  # 同理判断垂直移动的方向，就是用keys[pygame. pygame.K\_DOWN]-keys[ pygame.K\_UP]                  move\_ver = keys[pygame.K\_DOWN]-keys[ pygame.K\_UP]                  # 碰撞检测方法的调用                  self.check\_collide()                    # 控制逐帧动画播放速度                  frame\_counter = (frame\_counter + 1) % FRAME\_INTERVAL                  self.all\_group.update(frame\_counter == 0,move\_hor,move\_ver)  # 这里也需要传递移动方向              #绘制所有精灵              self.all\_group.draw(self.main\_window)              #刷新界面              pygame.display.update()              #设置游戏的刷新频率              clock.tick(60)      def event\_handler(self):          """事件监听"""          for event in pygame.event.get():              if event.type == pygame.QUIT: #处理点击窗口的关闭按钮的退出事件                  return True              elif event.type == pygame.KEYDOWN and event.key==pygame.K\_ESCAPE: # 按esc键程序也会退出                  return True              elif event.type == pygame.KEYDOWN and event.key==pygame.K\_SPACE: # 按空格键                  #如果游戏已经结束                  if self.is\_game\_over:                      self.reset\_game() #重置游戏                  else:                      self.is\_pause = not self.is\_pause # 如果没有结束，就可以进行暂停和继续的切换              # 当游戏没有暂停也没有结束，用户按下b键才有效，否则没有效果              # 引爆炸弹监听事件，用户按下b键会引爆一个炸弹把所有敌机都炸掉，然后炸弹数量要减少1              if not self.is\_game\_over and not self.is\_pause:                  if event.type == pygame.KEYDOWN and event.key==pygame.K\_b:                     score = self.hero.blowup(self.enemies\_group) # 炸弹屏幕上所有敌人并且获取总分                     # 更新面板的炸掉显示数量                     self.hud\_panel.show\_bomb(self.hero.bomb\_count)                     # 加分并且检查是否升级                     if self.hud\_panel.increase\_score(score):                         # 如果升级，需要创建更多敌机                         self.creat\_enemies()                  elif event.type == HERO\_DEAD\_EVENT:                      self.hud\_panel.lives\_count -= 1 # 每死亡一架飞机，生命数量减少1                      self.hud\_panel.show\_lives() # 更新生命标签的显示数据                      self.hud\_panel.show\_bomb(self.hero.bomb\_count) # 需要更新炸弹的显示数量                  elif event.type == HERO\_POWER\_OFF\_EVENT:                      self.hero.is\_power = False # 过滤无敌时间，响应将无敌标志设置为False                      # 然后需要关闭定时器                      pygame.time.set\_timer(HERO\_POWER\_OFF\_EVENT,0)                  elif event.type == HERO\_FIRE\_EVENT:                      self.hero.fire(self.all\_group)                  # 投放道具事件处理                  elif event.type == THROW\_SUPPLY\_EVENT:                      # print(len(self.supplies\_group.sprites()))                      supply = random.choice(self.supplies\_group.sprites()) # 随机获取一个道具                      supply.throw\_supply() # 把它投放出去                  elif event.type == BULLET\_ENHANCE\_OFF\_EVENT:                      # 1.把子弹的类改为普通子弹，也就是0                      self.hero.bullets\_kind = 0                      # 2.关闭子弹失效事件定时器                      pygame.time.set\_timer(BULLET\_ENHANCE\_OFF\_EVENT,0)          return False #其他情况都不会退出程序      def check\_collide(self):          """碰撞检测"""          # 根据游戏的规则，需要检测英雄飞机是否是在3秒的无敌时间内，如果不是，继续往下执行          # 检测英雄飞机和敌机方式碰撞          if not self.hero.is\_power:              # 获取于英雄飞机发生碰撞的敌机。是一个列表              collided\_enemies = pygame.sprite.spritecollide(self.hero,                                                          self.enemies\_group,False,                                                              pygame.sprite.collide\_mask)              # 过滤掉爆炸敌机残骸              collided\_enemies = list(filter(lambda x: x.hp > 0,collided\_enemies))              if collided\_enemies:                  self.hero.hp = 0 # 只要有碰撞，英雄飞机被摧毁              # 同时碰撞到的所有敌机也会被摧毁              for enemy in collided\_enemies:                  enemy.hp = 0              # 子弹和敌机的碰撞检测              #1.获取被撞击的敌机集合，是一个字典，key是敌机，value是击中敌机的字典列表              hit\_enemies = pygame.sprite.groupcollide(self.enemies\_group,self.hero.bullets\_group,                                                             False,False,pygame.sprite.collide\_mask)              # 2.遍历集合，判断有没有被摧毁的敌机              for enemy in hit\_enemies:                  # 如果是被摧毁的敌机，就不需要处理                  if enemy.hp <= 0:                      continue                  # 如果敌机没有被摧毁，遍历敌机对应的子弹列表                  for bullet in hit\_enemies[enemy]:                      bullet.kill() #销毁子弹                      enemy.hp -= bullet.damage                      if enemy.hp > 0: # 如果敌机没有被销毁，继续遍历下一颗子弹                          continue                      # 当前这一颗子弹以经把敌机摧毁了,加分并且判断是否升级                      if self.hud\_panel.increase\_score(enemy.value):                          #如果升级，需要创建更多敌机                          self.creat\_enemies()                      # 这架飞机已经被摧毁就不需要遍历下一颗子弹了                      break              # 英雄飞机拾取道具              supplies = pygame.sprite.spritecollide(self.hero,self.supplies\_group,False,                                                            pygame.sprite.collide\_mask)              if supplies:                  supply = supplies[0]                  # 将道具放到窗口下面                  supply.rect.y = SCREEN\_RECT.h                  # 判断道具类型                  if supply.kind == 0: # 是炸弹，需要增加屏幕上炸弹的数量和英雄飞机炸弹的数量                      self.hero.bomb\_count += 1                      self.hud\_panel.show\_bomb(self.hero.bomb\_count)                  else: # 是子弹增强效果                      # 把子弹类型改为双排                      self.hero.bullets\_kind = 1                      # 每隔20秒发送子弹增强效果关闭事件                      pygame.time.set\_timer(BULLET\_ENHANCE\_OFF\_EVENT,20000)  if \_\_name\_\_ == '\_\_main\_\_':      #初始化游戏      pygame.init()      #开始游戏      Game().start()      #释放游戏资源      pygame.quit() |

## game\_items.py

|  |
| --- |
| #封装游戏元素对应的类  from typing import Any  import pygame  import random  # 定义全局常量  SCREEN\_RECT = pygame.Rect(0,0,480,700)                                  #游戏屏幕矩形  FRAME\_INTERVAL = 10                                                     # 逐帧动画间隔帧数  HERO\_BOMB\_COUNT = 3                                                     # 英雄飞机的炸弹数量  HERO\_DEFAULT\_MID\_BUTTOM = (SCREEN\_RECT.centerx,SCREEN\_RECT.bottom - 90) #英雄默认初始位置  HERO\_DEAD\_EVENT = pygame.USEREVENT                                      # 定义英雄牺牲事件  HERO\_POWER\_OFF\_EVENT = pygame.USEREVENT+1                               # 定义英雄无敌数据结束事件  HERO\_FIRE\_EVENT = pygame.USEREVENT+2                                    #定义游戏飞机发射子弹的事件  THROW\_SUPPLY\_EVENT = pygame.USEREVENT+ 3                                #投放道具事件  BULLET\_ENHANCE\_OFF\_EVENT = pygame.USEREVENT+ 4                          #双排子弹失效事件  class GameSprite(pygame.sprite.Sprite):      res\_path = './res/images/'      def \_\_init\_\_(self,image\_name,speed, \*groups) -> None:          super().\_\_init\_\_(\*groups)          self.image = pygame.image.load(self.res\_path + image\_name) #加载图片          self.rect = self.image.get\_rect() # 获取图片的矩形区域并且保存起来          self.speed = speed # 接收传递进来的速度值          # 需要一个遮罩属性用来做高效碰撞检测          self.mask = pygame.mask.from\_surface(self.image) # 来用精灵的图片来创建遮罩      def update(self, \*args) -> None:          # super().update(\*args)          """默认在垂直方向移动"""          self.rect.y += self.speed  class BackgroundSprite(GameSprite):      # 需要在初始化的时候给他传递应该状态,is\_alt==True说明这个背景精灵需要显示在窗口的顶部      def \_\_init\_\_(self, is\_alt ,\*groups) -> None:          super().\_\_init\_\_('background.png', 1, \*groups)          if is\_alt:              self.rect.y = -self.rect.h      def update(self, \*args) -> None:          super().update(\*args)          # 图片滚出边界的处理：如果图片已经滚动到了窗口的底部，          # 立即让他回到窗口的最上面，然后重新往下滚动          if self.rect.y > self.rect.h:              self.rect.y = -self.rect.h  class StatusButton(GameSprite):      def \_\_init\_\_(self, image\_names, \*groups) -> None:          """image\_names是一个元组，元组的0下标必须是一张暂停的图片，元组的1下标必须是一张播放的图片"""          super().\_\_init\_\_(image\_names[0],0,\*groups)          # 准备2张图片供切换          self.images = [pygame.image.load(self.res\_path+name) for name in image\_names]      def switch\_status(self,is\_pause):          """切换状态的方法"""          #根据状态切换图片          self.image =self.images[1 if is\_pause  else 0] # 仿三目运算符  #标签精灵类  class Label(pygame.sprite.Sprite):      font\_path = './res/font/MarkerFelt.ttc' # 字体文件的存放路径      def \_\_init\_\_(self,text,size,color, \*groups) -> None:          super().\_\_init\_\_(\*groups)          # 创建字体，使用我们提供的字体文件          self.font = pygame.font.Font(self.font\_path,size)          self.color = color # 文字颜色          # 精灵属性          self.image = self.font.render(text,True,self.color)          self.rect = self.image.get\_rect()      def set\_text(self,text):          """设置字体方法"""          self.image = self.font.render(text,True,self.color)          self.rect = self.image.get\_rect()  # 飞机类  class Plane(GameSprite):      def \_\_init\_\_(self, hp,speed,value,wav\_name,normal\_names,hurt\_name,destroy\_names, \*groups) -> None:          """初始化方法          hp:            血量          speed:         飞行速度          value:         分值，也就是摧毁一个得到多少分          wav\_name:      对应的音效文件          normal\_names:  正常状态下的逐帧动画图片列表          hurt\_name:     受伤图片，此时飞机并没有被摧毁          destroy\_names: 飞机爆炸效果逐帧动画的图片列表          \*groups:       需要添加到的精灵组          """          super().\_\_init\_\_(normal\_names[0], speed, \*groups)          self.hp = hp          self.max\_hp = hp          self.value = value          self.wav\_name = wav\_name          # 正常状态图像列表          self.normal\_images = [pygame.image.load(self.res\_path+name) for name in normal\_names] #列表推导式          # 正常状态图像索引          self.normal\_index  = 0          # 加载受伤图片          self.hurt\_image = pygame.image.load(self.res\_path+hurt\_name)          # 加载摧毁图片列表          self.destroy\_images = [pygame.image.load(self.res\_path+name) for name in destroy\_names]          # 摧毁图片索引          self.destroy\_index = 0      # 重置飞机的方法      def reset\_plane(self):          #1.利用max\_hp重置hp          self.hp = self.max\_hp          #2.把normal\_index归零]          self.normal\_index = 0          #3.把destroy\_index归零          self.destroy\_index = 0          #4.把图片改为正常图片的第一张          self.image = self.normal\_images[0]      # update方法      def update(self,\*args):          """更新状态，准备下一次需要显示的内容"""          # 判断是否需要更新，只需要判断args[0]是否位True，注意\*args是可变长参数          if not args[0]: # 如果为False就不更新，直接return              return          #True就更新          #根据血量hp来切换需要显示的图片，有几种情况          if self.hp == self.max\_hp: # 说明没有受伤，显示正常图片即可              self.image = self.normal\_images[self.normal\_index]              # 计算下一张图片索引，注意下标不能越界              count = len(self.normal\_images)              self.normal\_index = (self.normal\_index + 1)%count # 利用取余数的方法来处理越界问题          elif self.hp >0 : # 受伤了，但是还没有被摧毁              self.image = self.hurt\_image #显示为受伤状态          else: #进入摧毁中的状态，播放摧毁动画              #判断是否播放到了最后一张图片，如果是就是完全被摧毁了              if self.destroy\_index < len(self.destroy\_images):                  self.image = self.destroy\_images[self.destroy\_index]                  self.destroy\_index += 1                else: #完全摧毁，需要重置飞机                  self.reset\_plane()  # 敌机类  class Enemy(Plane):      def \_\_init\_\_(self, kind,max\_speed, \*groups) -> None:          self.kind = kind          self.max\_speed = max\_speed          if kind == 0: # 小敌机              super().\_\_init\_\_(1, max\_speed, 1000, 'enemy1\_down.wav',                               ["enemy1.png"], "enemy1.png",                               ['enemy1\_down%d.png'% i for i in range(1,5)],                               \*groups)          elif kind == 1: # 中敌机              super().\_\_init\_\_(6, max\_speed, 6000, 'enemy2\_down.wav',                               ["enemy2.png"], "enemy2\_hit.png",                               ['enemy2\_down%d.png'% i for i in range(1,5)],                               \*groups)          elif kind == 2: # 大敌机              super().\_\_init\_\_(15, 1, 15000, 'enemy3\_down.wav',                               ["enemy3\_n1.png","enemy3\_n2.png"], "enemy3\_hit.png",                               ['enemy3\_down%d.png'% i for i in range(1,7)],                               \*groups)            # 随机设置飞机位置，要不然飞机就重叠在一起了          self.reset\_plane()        def reset\_plane(self):         super().reset\_plane() #先调用父类的重置方法         """敌机的数据重置代码"""         x = random.randint(0,SCREEN\_RECT.w - self.rect.w)         y = random.randint(0,SCREEN\_RECT.h - self.rect.h) - SCREEN\_RECT.h # 设置到屏幕外面         self.rect.topleft = (x,y)         #设置速度         self.speed = random.randint(1,self.max\_speed)      def update(self, \*args): #其实就是在父类的基础上添加飞机出界处理          super().update(\*args)   # 调用父类的update方法          #判断是否被摧毁，如果没有就继续更新位置          if self.hp > 0:              self.rect.y += self.speed          if self.rect.y >= SCREEN\_RECT.h: #出界了，就要重新设置位置              self.reset\_plane()  class Hero(Plane):      def \_\_init\_\_(self, \*groups) -> None:          self.is\_power = False                     # 是否无敌          self.bomb\_count = HERO\_BOMB\_COUNT         # 英雄飞机的炸弹数量          self.bullets\_kind = 0                      # 子弹类型          self.bullets\_group = pygame.sprite.Group() # 子弹精灵组          # 调用父类的初始化方法          super().\_\_init\_\_(1000, 5, 0, 'me\_down.wav',['me1.png','me2.png'],'me1.png',                      ['me\_destroy\_%d.png' % i for i in range(1,5)], \*groups)          # 设置英雄飞机默认位置          self.rect.midbottom = HERO\_DEFAULT\_MID\_BUTTOM          # 用定时器发送英雄发射子弹事件          pygame.time.set\_timer(HERO\_FIRE\_EVENT,200)      def update(self, \*args):          """          在这里args[0]是一个标记表明是否需要更新帧动画,          args[1]就是水平移动的方向基数，          args[2]就是垂直移动方向基数          """          super().update(\*args)          # 为了保证代码的正确执行，需要两个条件：首先传递进来的args的长度必须是3，其次英雄飞机的血量必须大于0          if len(args) != 3 or self.hp <= 0: #不符合条件就返回              return          # 根据args[1]来设置英雄飞机的水平移动方向          self.rect.x += args[1] \* self.speed          self.rect.x = 0 if self.rect.x < 0 else self.rect.x #左边出界处理          if self.rect.right > SCREEN\_RECT.right:               self.rect.right = SCREEN\_RECT.right # 右边出界处理          # 根据args[2]来设置英雄飞机的垂直移动方向          self.rect.y += args[2] \* self.speed          self.rect.y = 0 if self.rect.y < 0 else self.rect.y # 上边出界处理          if self.rect.bottom > SCREEN\_RECT.bottom:               self.rect.bottom = SCREEN\_RECT.bottom # 下边出界处理      def blowup(self,enemies\_group):          # 先判断有没有炸弹          if self.bomb\_count <= 0 or self.hp <=0:              return 0 # 没有炸弹， 炸不了,返回0分            self.bomb\_count -= 1  # 每炸一次，炸弹数需要减少1          score = 0          count = 0          # 炸掉使用敌机并且获取分数          for enemy in enemies\_group.sprites():              # 在炸飞机之前需要先判断它是否在屏幕里面,如果是才能炸              if enemy.rect.bottom > 0:                  score += enemy.value # 累计得分                  enemy.hp = 0 # 搞死敌人                  count += 1 # 计算器累计          print('炸毁了 %d 架敌机' % count)          # 返回获取的累计分数          return score        def reset\_plane(self):          super().reset\_plane()          self.is\_power = True          self.bomb\_count = HERO\_BOMB\_COUNT          self.bullets\_kind = 0            #发布英雄牺牲事件，让游戏主逻辑更新面板          pygame.event.post(pygame.event.Event(HERO\_DEAD\_EVENT))          # 给英雄飞机设置3秒的无敌时间          pygame.time.set\_timer(HERO\_POWER\_OFF\_EVENT,3000) # 超过三秒就没有无敌效果      # 英雄飞机发射子弹的方法      def fire(self,display\_group):          # 先要准备子弹需要添加到的精灵组          groups = (self.bullets\_group,display\_group)          # 测试子弹增强效果          # self.bullets\_kind = 1          # print("xxxxxxx")          # 创建子弹精灵          for i in range(3):              bullet1 = Bullet(self.bullets\_kind,\*groups)              #计算子弹垂直位置              y = self.rect.y - i\*15              #判断子弹类型              if self.bullets\_kind == 0: #单排                  bullet1.rect.midbottom = (self.rect.centerx,y)              else: #双排                   # 把第一排子弹的x坐标从中间往左边移动20像素                   bullet1.rect.midbottom = (self.rect.centerx - 20,y)                   #再创建一颗子弹                   bullet2 = Bullet(self.bullets\_kind,\*groups)                    # 把第二排子弹的x坐标从中间往右边边移动20像素                   bullet2.rect.midbottom = (self.rect.centerx + 20,y)    # 子弹类  class Bullet(GameSprite):      def \_\_init\_\_(self, kind, \*groups) -> None:          # image\_name = 'bullet.png' if kind==0 else 'bullet2.png'          image\_name = 'bullet1.png' if kind==0 else 'bullet2.png'          super().\_\_init\_\_(image\_name, -12, \*groups) # -12表示子弹往上移动，速度是12          self.damage = 1  #杀伤力      def update(self, \*args) -> None:          super().update(\*args)          # 子弹飞出窗口的处理，需要销毁          if self.rect.bottom < 0:              self.kill()   # 调用精灵自身的kill方法来销毁它  # 道具类，可以增加炸弹数量或者把子弹变为双排，维持20秒钟  class Supply(GameSprite):      def \_\_init\_\_(self, kind, \*groups) -> None:          # 根据道具类型的不同加载不同图片文件          image\_name = "bomb\_supply.png" if kind == 0 else 'bullet\_supply.png'          super().\_\_init\_\_(image\_name, 5, \*groups)          self.kind = kind          # 根据道具类型的不同加载不同音效文件          self.wav\_name = "get\_%s.wav" % ("bomb" if kind == 0 else "bullet")            # 根据游戏规则，创建完道具对象后，需要把他们放在游戏窗口的下方          self.rect.top = SCREEN\_RECT.h      def update(self, \*args) -> None:          # 需要先判断，如果道具已经在窗口下方，不需要更新          if self.rect.y > SCREEN\_RECT.h:              return          super().update(\*args)        # 抛出道具的方法      def throw\_supply(self):          self.rect.bottom = 0 # 将道具移动到窗口顶部          # 计算道具随机出现的位置的x左边          self.rect.x = random.randint(0,SCREEN\_RECT.w - self.rect.w) |

## game\_hud.py

|  |
| --- |
| #游戏面板模块  import pygame  from game\_items import \*  #面板类  class HUDPanel(object):      margin =10      white =(255,255,255)      gray = (64,64,64)      reward\_score = 100000  #每增加100000分奖励一架飞机      level2\_score = 10000     #达到10000分就进入第二关      level3\_score = 50000     #达到50000分就进入第三关      record\_filename = 'record.txt' # 保存最高分数的文件的路径      def \_\_init\_\_(self,display\_group) -> None:          """          面板初始化方法：          display\_group是面板中的精灵要被添加到的精灵组          """          self.score = 0          self.lives\_count = 3          self.level = 1          self.best\_score = 0          # 加载上一次游戏的最好成绩          self.load\_best\_score()            # 图像精灵          #1.状态精灵          self.status\_sprite = StatusButton(('pause\_nor.png','resume\_nor.png'),display\_group)          self.status\_sprite.rect.topleft = (self.margin,self.margin)          #2.炸弹精灵          self.bomb\_sprite = GameSprite('bomb.png',0,display\_group)          # 调整bomb\_sprite的位置          self.bomb\_sprite.rect.x = self.margin          self.bomb\_sprite.rect.bottom = SCREEN\_RECT.bottom - self.margin          #3.生命计数精灵          self.lives\_sprite = GameSprite('life.png',0,display\_group)          # szlives\_sprite的位置          self.lives\_sprite.rect.right = SCREEN\_RECT.right-self.margin          self.lives\_sprite.rect.bottom = SCREEN\_RECT.bottom - self.margin          #4.创建标签精灵          # 4.1数字标签          # 4.1.1 分数标签          self.score\_label = Label("%d" % self.score,32,self.gray,display\_group)          self.score\_label.rect.midleft = (self.status\_sprite.rect.right + self.margin,                                             self.status\_sprite.rect.centery)            # 4.1.2 炸弹计数标签          self.bomb\_label = Label("X 3",32,self.gray,display\_group )          self.bomb\_label.rect.midleft = (self.bomb\_sprite.rect.right+self.margin,                                             self.bomb\_sprite.rect.centery)          # 4.1.3 生命值标签,这个标签的位置笔记特别，它会占用lives\_sprite的位置，          # 然后lives\_sprite的位置需要使用它的位置来重新计算          self.lives\_label = Label("X %d" % self.lives\_count,32,self.gray,display\_group)          self.lives\_label.rect.midright = (SCREEN\_RECT.right - self.margin,                                            self.bomb\_label.rect.centery)            #修改 lives\_sprite的位置          self.lives\_sprite.rect.right = self.lives\_label.rect.left - self.margin          #注意：在游戏正常运行时，最好成绩标签，状态标签和  提示标签是隐藏的，游戏状态，他们才显示          # 4.1.4 最好成绩标签          self.best\_label = Label("Best: %d" % self.best\_score,36,self.white)            # 4.1.5 状态标签          self.status\_label = Label("Game Playing",48,self.white)            # 4.1.6 提示标签          self.tip\_label = Label("Press SpaceBar To Pause Game",22,self.white)          # 显示炸弹数量的方法      def show\_bomb(self,count):          # 修改文字          self.bomb\_label.set\_text("X %d" % count)          # 修改位置          self.bomb\_label.rect.midleft = (self.bomb\_sprite.rect.right + self.margin,                                          self.bomb\_sprite.rect.centery)        def show\_lives(self):          self.lives\_label.set\_text("X %d" % self.lives\_count)          #调整 lives\_label的位置          self.lives\_label.rect.midright = (SCREEN\_RECT.right - self.margin,                                            self.bomb\_label.rect.centery)            #调整 lives\_sprite的位置          self.lives\_sprite.rect.right = self.lives\_label.rect.left - self.margin      # 定义一个增加分数的方法      def increase\_score(self,enemy\_score):          #1.先用一个变量来累计每一次的得分          score = self.score + enemy\_score # 这个分数需要一直增加          #2.每累计一次，就需要判断是否奖励一条命，是很有技巧的,其实还有别的方法，如：只要这个score能被100000整除就可以了          if score // self.reward\_score !=self.score // self.reward\_score:             self.lives\_count +=1 #奖励一条命，也就是多一架飞机             self.show\_lives() # 获取到奖励后需要更新生命值          # 更新自己的分数          self.score = score          #3.每累计一次，都要和上一次的最好成绩比较,只要这个成绩比原来的最好成绩大，就把它设置为最好成绩，否则不设置          self.best\_score = score if score > self.best\_score else self.best\_score          #4.每累计一次，都要判断是否升级          if  score < self.level2\_score:              level = 1          elif score < self.level3\_score:              level = 2          else:              level = 3          is\_upgrade = level !=self.level          self.level = level          # 5.每累计一次，都需要修改分数标签的内容和位置          self.score\_label.set\_text("%d" % self.score)          self.score\_label.rect.midleft = (self.status\_sprite.rect.right + self.margin,                                             self.status\_sprite.rect.centery)          return is\_upgrade        # 保存最好成绩的方法      def save\_best\_score(self):          with open(self.record\_filename,'w') as f:              f.write('%d' % self.best\_score)      def load\_best\_score(self):           try:              with open(self.record\_filename,'r') as f:                  self.best\_score = int(f.read())                  print(f'上一次最高分数是{self.best\_score}')           except (FileNotFoundError,ValueError):               print('读取record.txt文件出错,文件不存在或者内容为空...')        # 暂停游戏的方法,必须在游戏没有介绍的前提下才有效      def panel\_pause(self,is\_game\_over,display\_group):          #1.首先需要判断是否把最好成绩标签，状态标签和  提示标签添加到了精灵组，如果是，直接返回          if display\_group.has(self.best\_label,self.status\_label,self.tip\_label):              return          # 2.根据游戏是否介绍来调整标签的文字          text = "Game Over!" if is\_game\_over else "Game Paused!"          tip = 'Press SpaceBar To'          tip += ' Play Again' if is\_game\_over else " Continue"          # 3.给标签设置对应的文字          self.status\_label.set\_text(text)          self.best\_label.set\_text("Best Score: %d" % self.best\_score)          self.tip\_label.set\_text(tip)          # 4.调整标签位置          self.best\_label.rect.center = SCREEN\_RECT.center          self.status\_label.rect.midbottom = (self.best\_label.rect.centerx,                                              self.best\_label.rect.y - 2 \* self.margin)            self.tip\_label.rect.midtop = (self.best\_label.rect.centerx,                                      self.best\_label.rect.bottom + 8 \* self.margin)            # 5.添加标签到精灵组          display\_group.add(self.best\_label,self.status\_label,self.tip\_label)          # 6.状态精灵切换状态          self.status\_sprite.switch\_status(True)      # 从暂停状态恢复      def panel\_resume(self,display\_group):          # 1.隐藏 最好成绩标签，状态标签和  提示标签添只需要把他们从精灵组里面删除即可          display\_group.remove(self.best\_label,self.status\_label,self.tip\_label)          # 2.需要切换状态精灵的状态          self.status\_sprite.switch\_status(False)      # 重置面板内容的功能      def reset\_panel(self):          #重置数据          self.score = 0          self.lives\_count = 3          #重置精灵数据          self.increase\_score(0) # 可以在这个方法里面计算level，设置最好成绩标签，状态标签和  提示标签添等等          self.show\_lives() #重置生命值          self.show\_bomb(3) #重置炸弹数量 |

## game\_music.py 暂时为空，我们下一节会完善它的代码