# 这一节我们来学习指示面板的制作

# 1.面板功能介绍p28

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

# 2.面板类的属性和方法说明p29

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

# 3.面板类初始化p30

|  |
| --- |
|  |

## 1.我们在game\_hud.py新建一个面板类，实现初始化方法

|  |
| --- |
|  |

# 4.状态按键精灵的定义p31

## 1.类设计图

|  |
| --- |
|  |

## 2.代码实现，在game\_item.py模块里面创建一个StatusButton类继承GameSprite

|  |
| --- |
|  |

# 5.显示并定位状态按钮、炸弹精灵、生命精灵p32

## 1.创建图像精灵

|  |
| --- |
|  |
| #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)      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          # 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)          #3.生命计数精灵          self.lives\_sprite = GameSprite('life.png',0,display\_group)          # 数字标签          # 提示标签 |

## 2. 在game.py模块中创建游戏控制面板

|  |
| --- |
|  |

## 3.测试一下，精灵是有了，但是位置不正确

|  |
| --- |
|  |

## 4.我们来调整他们的位置，回到game\_hud.py模块中，修改各个精灵的位置

|  |
| --- |
|  |

## 5、测试一下位置基本正确

|  |
| --- |
|  |

# 6.标签类功能说明p33

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

# 7.定义标签精灵类p34

## 我们在game\_items.py模块里面创建一个标签精灵类，代码如下

|  |
| --- |
|  |

### 注意：这个类不需要经常更新，所以它没有重写update方法，需要更新内容的时候调用自身的set\_text方法

# 8.显示数字标签p35

|  |
| --- |
|  |
|  |
|  |
|  |
| **坐标的中心对齐问题** |
| #游戏面板模块，本阶段的game\_hud.py模块的代码如下  import pygame  from game\_items import \*  #面板类  class HUDPanel(object):      margin =10      white =(255,255,255)      gray = (64,64,64)      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          # 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          # 4.1.4 最好成绩标签          self.best\_label = Label("Best: %d" % self.best\_score,36,self.white,display\_group)          self.best\_label.rect.center = SCREEN\_RECT.center          # 4.1.5 状态标签          self.status\_label = Label("Game Over",48,self.white,display\_group)          self.status\_label.rect.midbottom = (self.best\_label.rect.centerx,                                              self.best\_label.rect.y - 2 \* self.margin)          # 4.1.6 提示标签          self.tip\_label = Label("Press SpaceBar To Play Again",22,self.white,display\_group)          self.tip\_label.rect.midtop = (self.best\_label.rect.centerx,                                        self.best\_label.rect.bottom + 8 \* self.margin)          # 4.2提示标签 |

## 测试一下，是能够显示所有数字标签的标签位置都正确

|  |
| --- |
|  |

# 9.显示提示信息标签p36

## 这个上面已经做了。只需要听一听

# 10修改炸弹计数p37

|  |
| --- |
|  |
|  |

## 1.在game\_hud.py模块中面板类添加show\_bomb方法

|  |
| --- |
|  |

## 2.在game.py中添加对按下b键的事件监听，暂时只是用随机数测试一下是否能够响应，代码以后要修改

|  |
| --- |
|  |

## 测试一下，能够响应

|  |
| --- |
|  |
|  |

# 11.修改生命值计数p38

|  |
| --- |
|  |

## 我们在game\_hud.py模块里面编写show\_bomb方法和show\_lives方法

|  |
| --- |
|  |

## 然后在game.py模块中利用按键b的监听事件来测试这两个函数，如果代码没有问题是可以通过测试的

|  |
| --- |
|  |

## 此时按b（在英文输入法状态下）炸弹是数量会改变并且显示的位置不变

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

## 好我们来此时生命值并且

|  |
| --- |
|  |

## 效果如下

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

### 可以看出这两个函数代码没有问题

# 12.修改游戏得分的算法p39

## 描述

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

## is\_upgrade是用来改变升级后的场景的

## 在字母b键的监听函数里面测试一下，效果如图

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

## 说明这个加分方法可以正常运行

# 13.游戏得分更新p40

## 需要在game.py模块中添加测试代码

|  |
| --- |
|  |

也是ok的

# 14，最好成绩文本读取说明p41

## 思路说明

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

# 15.保存最好成绩到文件p42

## 代码实现：

## 1.先在game.py的同一级目录下面新建一个record.txt,用来保存最好成绩

|  |
| --- |
|  |

## 2.在geme\_hud.py在添加一个保存最好成绩的方法save\_best\_score

|  |
| --- |
|  |
|  |

## 3.需要在游戏退出之前调用save\_best\_score方法来保存最好成绩，在game.py文件中添加如下代码

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

### 说明这个方法工作正常

# 16.从文件在加载最好成绩p43

## 既然可以在游戏介绍前保存最高分数，那么必然也需要在游戏加载是获取上一次介绍时保存的最高分数，我们给HUDPanel类编写一个load\_best\_score方法，使他with open语句比较好，它会自动关闭文件句柄在HUDPanel类的初始化方法的最后调用最高读取最高分数的方法

|  |
| --- |
|  |

## 这里有一个问题：读取文件的时候可能文件不存在，我们需要添加对文件不存在异常和文件内容为空异常(ValueError)处理，修改load\_best\_score方法的代码如下

|  |
| --- |
|  |

### 这样子处理后代码就比较健壮了

|  |
| --- |
|  |

# 17.为什么只会保存最高得分？p44

|  |
| --- |
|  |

# 18游戏状态变化的说明p45

|  |
| --- |
|  |

# 19精灵的绘制顺序演示p46

|  |
| --- |
|  |
|  |

# 20. 处理提示信息和隐藏的思路p47

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

### 可见如果精灵不添加到精灵组，它是不可见的，因为精灵组的绘制方法中没有办法绘制它

# 21.显示提示信息的代码实现p48

## 说明

|  |
| --- |
|  |
|  |

## 代码实现

## 1.把最好成绩标签，状态标签和  提示标签隐藏，因为在游戏正常运行时，他们是不可见的，方法是创建的时候不要添加到精灵组

|  |
| --- |
|  |

## 2.给HUDPanel类添加一个暂停方法panel\_pause，在这个方法里面需要显示上面三个标签，并且修改一下状态

|  |
| --- |
|  |

## 3.测试一下，在game.py模块中的事件处理函数里面代码下面的代码

|  |
| --- |
|  |

### 效果：

|  |
| --- |
|  |
|  |

### 说明这个暂停方法没有问题

# 22. 隐藏提示信息的处理p49

## 需要给HUDPanel类添加一个从暂停状态切换到继续游戏的状态的方法panel\_resume

|  |
| --- |
|  |
|  |

## 代码实现

## 1.进入game\_hud.py模块，添加panel\_resume方法

|  |
| --- |
|  |

## 2.测试，在事件处理代码else出添加测试代码

|  |
| --- |
|  |

### 效果：

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

### 说明这个方法是能够正常工作的

# 23.为什么要重置面板数据的说明p50

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

## 虽然能够实现功能但是还有有点问题，我们需要重置面板的内容

# 24.重置面板数据的代码实现p51

## 1.给HUDPanel类添加一个重置面板的功能：reset\_panel

|  |
| --- |
|  |

## 2.然后在game.py模块里面Game类的reset\_game方法里面调用这个重置面板的方法

|  |
| --- |
|  |

## 3.测试

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

### 说明这个功能也是正常的

# 这一节就到这里，这一节完成后的各个模块的代码如下

## 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))             #创建英雄精灵，静止不动         hero = GameSprite('me1.png',0,self.all\_group)         hero.rect.center = SCREEN\_RECT.center #居中显示         #创建游戏控制面板         self.hud\_panel = HUDPanel(self.all\_group)         #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      def creat\_enemies(self):          """创建敌机"""          pass      def creat\_supplies(self):          """创建道具"""          pass      def start(self):          """开始游戏"""          #创建游戏时钟          clock = pygame.time.Clock()          #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                  if self.hud\_panel.increase\_score(10): # 模拟得分测试代码                      print(f'升级到{self.hud\_panel.level}')                    self.all\_group.update()              #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:                      # 引爆一颗炸弹，以后实现                      #并且炸弹的计数要减少1                      self.hud\_panel.show\_bomb(random.randint(0,100)) #不是随机的，这里只是测试一下会不会响应                      # 模拟英雄牺牲                      self.hud\_panel.lives\_count -= 1                      self.hud\_panel.show\_lives()              return False #其他情况都不会退出程序      def check\_collide(self):          """碰撞检测"""          pass    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  # 定义全局常量  #1.游戏主窗口区域  SCREEN\_RECT = pygame.Rect(0,0,480,700)    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      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() |

## 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 :暂时没有内容