|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  | **河南工业大学期末作品成绩评价表** |  |  |
|  |  |  |  |  |  |
|  |  |  | **（2020~2021学年第1学期）** |  |  |
|  |  |  |  |  |  |
|  |  |  | 课程：软件体系结构 |  |  |
|  |  |  |  |  |  |
| 专业班级 | 软件1702 | 学号 | 201716040224 | 姓名 | 刘文博 |
| 成绩 评定 | 评分项 | 比例 | 得分 | | |
| 系统 | 20 | 系统运行情况（10） |  |  |
| 演示介绍情况（10） |  |
| 设计 | 40 | 基于架构模式的设计能力（20） |  |  |
| 架构质量方面的设计考量（20） |  |
| 代码 | 40 | 架构质量属性实现能力（20） |  |  |
| 架构和设计模式实现能力（20） |  |
| 合计 | 100 |  | | |
|  | 综合评定为： | |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  | 日 期： |  |

 未标题-1

**软件体系结构**

**作品设计说明书**

**题 目: 坦克对战游戏**

**院系名称： 信息学院 专业班级： 软件1702**

**学生姓名： 刘文博 学 号： 201716040224**

**教师姓名： 刘 灿 课程名称： 软件体系结构**

**一．需求**

**功能性需求：**

**1.对功能的规定**

游戏包括的范围：本游戏包括主类、坦克类、子弹类、爆炸类、墙类、音效类。

要求游戏能有效、快速、安全、可靠和无误的完成上述操作。并要求客户端 的界面要简单明了、易于操作，服务器程序利于维护。

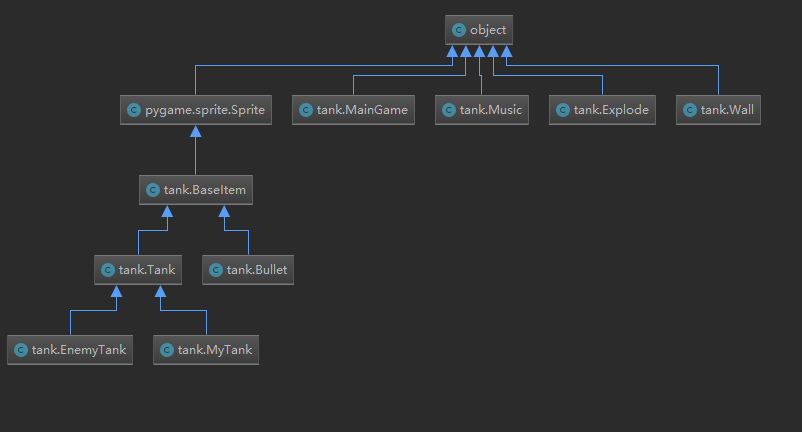
**2.对游戏操作的要求**

基本要求：坦克无法穿过坦克，坦克无法穿过墙壁，地方子弹碰到敌方坦克不做反应，地方子弹碰到我方坦克。我方坦克碰到敌方子弹，坦克会被击毁，子弹可以对墙壁造成伤害，当墙壁血量值为0时，墙壁消失。

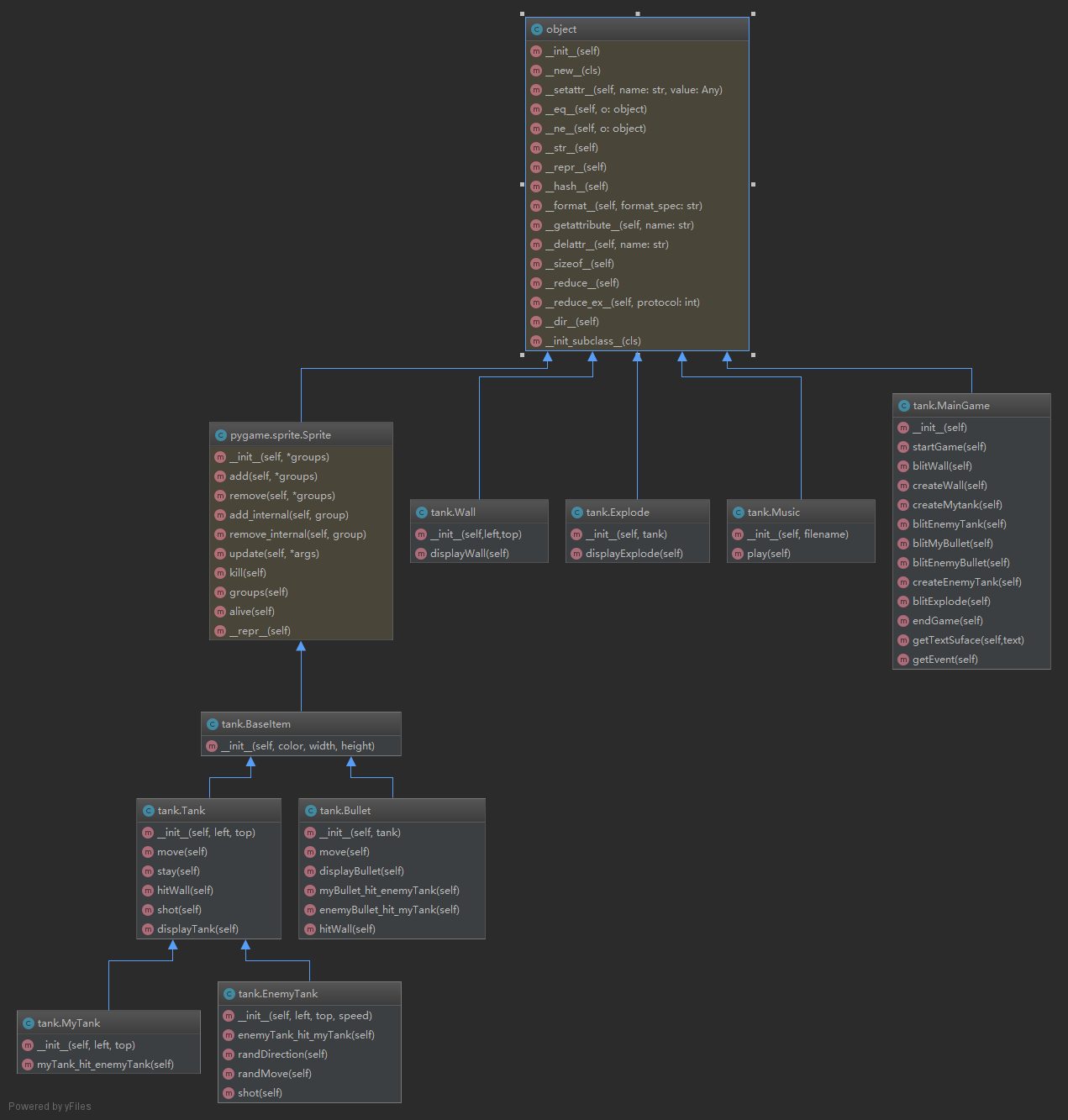
**3.程序包含的模块以及方法**

|  |
| --- |
| *坦克大战游戏的需求 1.项目中有哪些类 2.每个类中有哪些方法  1.坦克类（我方坦克、敌方坦克）  射击  移动类  显示坦克的方法  坦克是否与敌方坦克碰撞  坦克是否与墙碰撞 2.子弹类  移动  显示子弹的方法  我方子弹是否射中敌方坦克  地方子弹是否射中我方坦克  子弹是否射中墙壁 3.墙壁类  属性：是否可以通过 4.爆炸效果类  展示爆炸效果 5.音效类  播放音乐 6.主类  开始游戏  结束游戏  显示敌方子弹  显示我方子弹  显示爆炸效果  显示敌方坦克，显示我方坦克  显示墙壁  生成敌方坦克  生成我方坦克  生成墙壁  获取键盘事件* |

**模块之间的关系：**



**模块中的方法：**

****

**非功能性需求：**

1. **规范性**

开发过程控制、开发技术、系统编码、文档应规范化，并遵循

相应的国内外标准。开发结束，需要提供必要的文档资料。

1. **可靠性**

保证系统的可靠运行和在升级过程中的方便快捷。

1. **可扩充性**

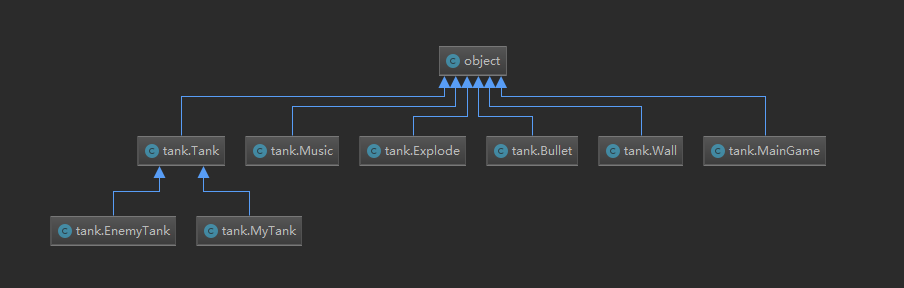
系统应当可以根据需求的变化，方便地进行功能的调整、增减，模块的升级和系统架构的逐步完善。

1. **界面友好、操作方便**

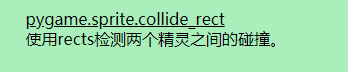
操作界面要直观、简单、贴近实际，操作过程应当尽量简化，符合实际过程。身份认证过程即要保证安全，也要尽量简化认证过程。

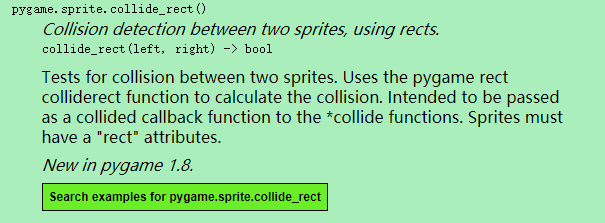
**二. 架构**

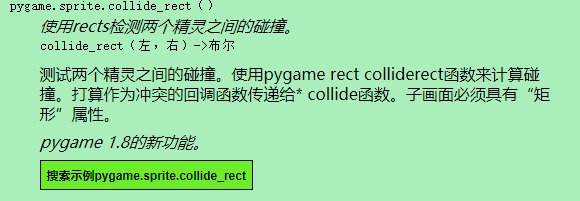
最开始的系统架构，只考虑了类与类之间的关系，设计如下：



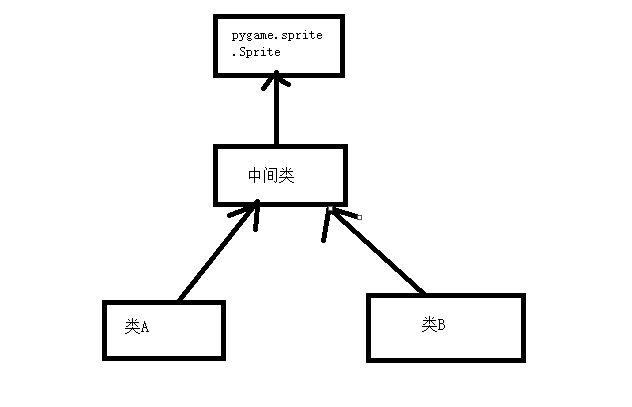
刚开始按照这种架构进行编程，没有问题，但是当项目需要解决穿过问题（只把图片加载进去时，坦克与坦克之间，坦克与墙壁之间，坦克与子弹之间可与穿过去）的时候，发现还需要解决物体之间的碰撞问题，如果自己手动实现通过图片的坐标计算，判断物体之间是否碰撞，这样就大大的增大了项目的难度，经过查看文档，发现pygame已经有实现好的类来解决这个问题，那就是通过实现或者继承pygame.sprite.Sprite（后面称为精灵类）这个类来实现判断物体之间是否发生碰撞。



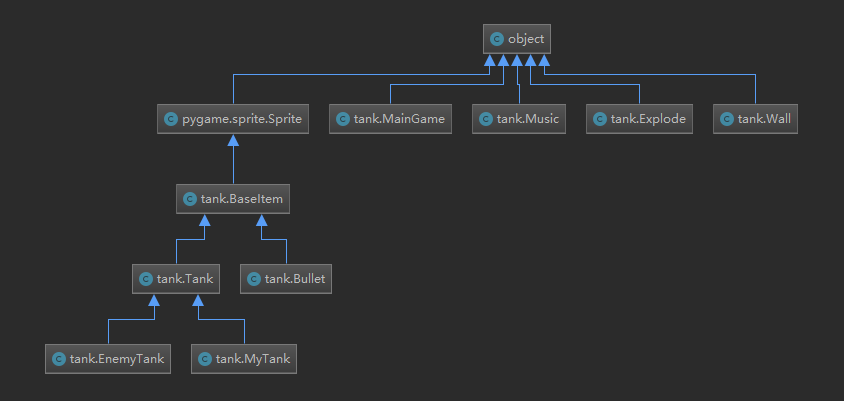




但是目前的问题是，想要使用精灵类的方法，就需要实现或者继承精灵类，在面向对象这门课程我们学过当有多个类都需要继承某一个类来使用某一个方法的时候，可以通过一个中间类，让中间类继承需要使用到的类，然后再将自己编写的类继承自中间类，这样就可以让自己写的类来使用想要的方法或者属性，这样做的好处是：当发现项目还需要更多的类库中的不同的方法的时候，只需要在中间类的实现上添加上目标类库，自己编写的类只需要继承自中间类，而不需要继承那么多的类，实现了低耦合这一特性，同样也会使得程序的扩展性大大增加，而不需要为了增加某一个功能改动全部的对象。

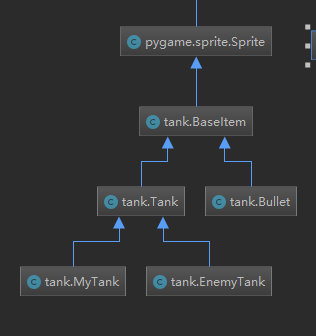


经过分许后，将类与类之间的关系进行了调整优化，调整后的系统架构为

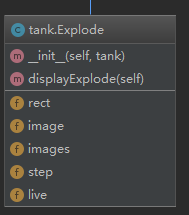


将坦克类与子弹类都继承自中间变量BaseItem这个类，就可以实现判断坦克与坦克之间，坦克与墙壁之间，坦克与子弹等之间是否发生碰撞，从而解决不同物体之间可以相互穿过的问题。

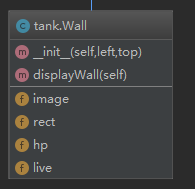
**三. 实现**



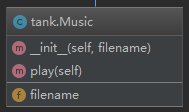
|  |
| --- |
| # 定义一个基类  class BaseItem(Sprite):  def \_\_init\_\_(self, color, width, height):  # Call the parent class (Sprite) constructor  pygame.sprite.Sprite.\_\_init\_\_(self)  class Tank(BaseItem):  # 添加距离左边left 距离上边top  def \_\_init\_\_(self, left, top):  self.images={  "U":pygame.image.load("img/p1tankU.gif"),  "D":pygame.image.load("img/p1tankD.gif"),  "L":pygame.image.load("img/p1tankL.gif"),  "R":pygame.image.load("img/p1tankR.gif")  }  # 方向  self.direction = "U"  # 根据当前图片的方向获取图片  self.image = self.images[self.direction]  # 根据图片获取区域  self.rect = self.image.get\_rect()  # 设置区域的left 和 top  self.rect.left = left  self.rect.top = top  # 速度 决定坦克移动的快慢  self.speed = 10  # 坦克移动的开关  self.stop = True  # 标记坦克是否活着  self.live = True  # 新增属性原来坐标  self.oldLeft=self.rect.left  self.oldTop=self.rect.top  # 移动  def move(self):  # 移动后记录原始的坐标  self.oldLeft = self.rect.left  self.oldTop = self.rect.top  # 判断坦克的方向进行移动  # 移动到边界就无法移动了  if self.direction == 'L':  if self.rect.left > 0:  self.rect.left -= self.speed  elif self.direction == 'U':  if self.rect.top > 0:  self.rect.top -= self.speed  elif self.direction == 'D':  if self.rect.top + self.rect.height < SCREEN\_HEIGHT:  self.rect.top += self.speed  elif self.direction == 'R':  if self.rect.left + self.rect.height < SCREEN\_WIDTH:  self.rect.left += self.speed  # 坦克原地静止  def stay(self):  self.rect.left = self.oldLeft  self.rect.top = self.oldTop  # 检测坦克是否与墙壁发生碰撞  def hitWall(self):  for wall in MainGame.wallList:  if pygame.sprite.collide\_rect(self, wall):  # 将坐标设置为移动之前的坐标  self.stay()  #射击  def shot(self):  return Bullet(self)  # 展示坦克的方法  def displayTank(self):  # 获取展示的对象  self.image = self.images[self.direction]  # 调用blit方法展示  MainGame.window.blit(self.image,self.rect)  # 敌方坦克  class EnemyTank(Tank):  def \_\_init\_\_(self, left, top, speed):  # 调用父类的初始化方法  super(EnemyTank, self).\_\_init\_\_(left, top)  # 加载图片集  self.images={  "U": pygame.image.load("img/enemy1U.gif"),  "D": pygame.image.load("img/enemy1D.gif"),  "L": pygame.image.load("img/enemy1L.gif"),  "R": pygame.image.load("img/enemy1R.gif")  }  # 方向 随机生成敌方坦克的方向  self.direction = self.randDirection()  # 根据方向获取坦克  self.image = self.images[self.direction]  # 获取到区域  self.rect = self.image.get\_rect()  self.rect.left = left  self.rect.top = top  self.speed = speed  # 移动开关  self.stop = True  # 新增一个步数变量  self.step = 20  # 敌方坦克与我方坦克是否发生碰撞  def enemyTank\_hit\_myTank(self):  if pygame.sprite.collide\_rect(self, MainGame.my\_tank):  self.stay()  # 随机生成敌方坦克的方向  def randDirection(self):  num = random.randint(1,4)  if num == 1:  return 'U'  elif num == 2:  return 'D'  elif num == 3:  return "L"  elif num == 4:  return 'R'  # 敌方坦克随机移动的方法  def randMove(self):  if self.step <= 0:  self.direction = self.randDirection()  self.step = 30 # 当一定步数后，重新开始移动  else:  self.move()  self.step -= 1  # 重写shot()  def shot(self):  # 随机生成150以内的数  num=random.randint(1,150)  if num<10:  return Bullet(self)  # 我方坦克  class MyTank(Tank):  def \_\_init\_\_(self, left, top):  super(MyTank, self).\_\_init\_\_(left, top)  # 检测我方坦克与敌方坦克发生碰撞  def myTank\_hit\_enemyTank(self):  # 循环遍历敌方坦克列表  for enemyTank in MainGame.enemyTankList:  if pygame.sprite.collide\_rect(self, enemyTank):  self.stay()  # 子弹类  class Bullet(BaseItem):  def \_\_init\_\_(self, tank):  self.image = pygame.image.load("img/enemymissile.gif")  # 坦克的方向决定子弹的方向  self.direction = tank.direction  # 获取到区域  self.rect = self.image.get\_rect()  # 子弹的位置与坦克的位置有关  if self.direction == 'U':  self.rect.left = tank.rect.left + tank.rect.width / 2 - self.rect.width / 2  self.rect.top = tank.rect.top - self.rect.height  elif self.direction == 'D':  self.rect.left = tank.rect.left + tank.rect.width / 2 - self.rect.width / 2  self.rect.top = tank.rect.top + tank.rect.height  elif self.direction == 'L':  self.rect.left = tank.rect.left - self.rect.width / 2 - self.rect.width / 2  self.rect.top = tank.rect.top + tank.rect.width / 2 - self.rect.width / 2  elif self.direction == 'R':  self.rect.left = tank.rect.left + tank.rect.width  self.rect.top = tank.rect.top + tank.rect.width / 2 - self.rect.width / 2  # 子弹的速度  self.speed = 6  self.live = True  # 移动  def move(self):  if self.direction == 'U':  if self.rect.top > 0:  self.rect.top -= self.speed  else:  self.live = False  elif self.direction == 'R':  if self.rect.left + self.rect.width < SCREEN\_WIDTH:  self.rect.left += self.speed  else:  self.live = False  elif self.direction == 'D':  if self.rect.top + self.rect.height < SCREEN\_HEIGHT:  self.rect.top += self.speed  else:  self.live = False  elif self.direction == 'L':  if self.rect.left > 0:  self.rect.left -= self.speed  else:  self.live = False  # 展示子弹的方法  def displayBullet(self):  # 将图片surface加载到窗口  MainGame.window.blit(self.image,self.rect)  # 我方子弹与敌方坦克的碰撞  def myBullet\_hit\_enemyTank(self):  # 循环遍历敌方坦克列表，判断是否发生碰撞  for enemyTank in MainGame.enemyTankList:  if pygame.sprite.collide\_rect(enemyTank, self):  # 修改敌方坦克和我方子弹的状态  enemyTank.live = False  self.live = False  # 创建爆炸对象  explode = Explode(enemyTank)  # 将爆炸对象添加到爆炸列表中  MainGame.explodeList.append(explode)  # 敌方子弹与我方坦克的碰撞  def enemyBullet\_hit\_myTank(self):  if MainGame.my\_tank and MainGame.my\_tank.live:  if pygame.sprite.collide\_rect(MainGame.my\_tank, self):  # 产生爆炸对象  explode = Explode(MainGame.my\_tank)  # 将爆炸对象添加到爆炸列表中  MainGame.explodeList.append(explode)  # 修改敌方子弹与我方坦克的状态  self.live = False  MainGame.my\_tank.live = False  # 子弹是否碰撞墙壁  def hitWall(self):  # 循环遍历墙壁列表  for wall in MainGame.wallList:  if pygame.sprite.collide\_rect(self, wall):  # 修改子弹的生存状态，让子弹消失  self.live = False  # 墙壁的生命值减小  wall.hp -= 1  if wall.hp <= 0:  # 修改墙壁的生存状态  wall.live = False |



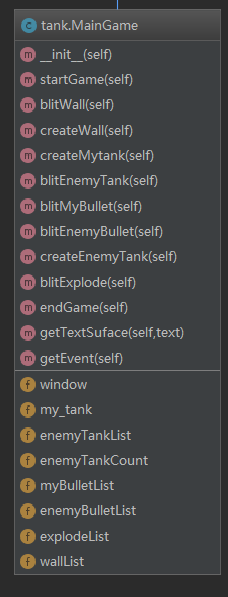
|  |
| --- |
| class Explode:  def \_\_init\_\_(self, tank):  # 爆炸的位置由当前子弹打中的坦克位置决定  self.rect=tank.rect  self.images=[  pygame.image.load('img/blast0.gif'),  pygame.image.load('img/blast1.gif'),  pygame.image.load('img/blast2.gif'),  pygame.image.load('img/blast3.gif'),  pygame.image.load('img/blast4.gif'),  ]  self.step = 0  self.image = self.images[self.step]  # 是否显示，爆炸完成后，应该消失  self.live = True  # 展示爆炸效果的方法  def displayExplode(self):  if self.step < len(self.images):  # 根据索引获取爆炸对象  self.image = self.images[self.step]  self.step += 1  # 添加到主窗口  MainGame.window.blit(self.image, self.rect)  else:  # 修改是否显示的状态  self.live = False  self.step = 0 |



|  |
| --- |
| class Wall:  def \_\_init\_\_(self,left,top):  # 加载墙壁图片  self.image = pygame.image.load('img/steels.gif')  # 获取墙壁的区域  self.rect = self.image.get\_rect()  # 设置位置left、top  self.rect.left = left  self.rect.top = top  # 是否存活  self.live = True  # 设置生命值  self.hp = 5  # 展示墙壁的方法  def displayWall(self):  MainGame.window.blit(self.image,self.rect) |



|  |
| --- |
| class Music():  def \_\_init\_\_(self, filename):  self.filename = filename  # 初始化音乐混合器  pygame.mixer.init()  # 加载音乐  pygame.mixer.music.load(self.filename)  # 播放音乐  # 播放音乐  def play(self):  pygame.mixer.music.play() |

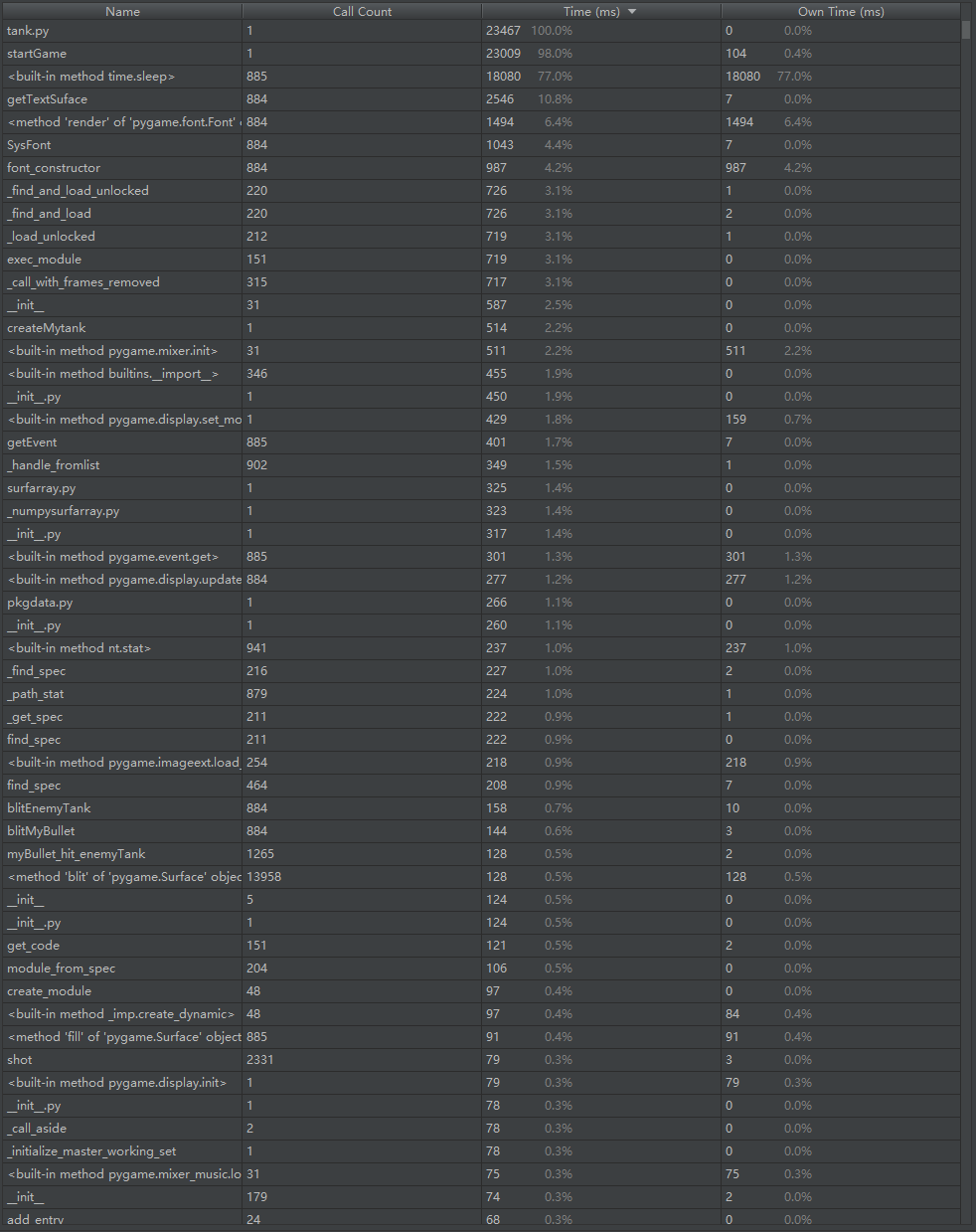


|  |
| --- |
| class MainGame():  window = None  my\_tank = None  # 存储敌方坦克的列表  enemyTankList = []  # 定义敌方坦克的数量  enemyTankCount = 5  # 存储我方子弹的列表  myBulletList = []  # 存储敌方子弹的列表  enemyBulletList = []  # 存储爆炸效果的列表  explodeList = []  # 存储墙壁的列表  wallList=[]  def \_\_init\_\_(self):  pass  # 开始游戏  def startGame(self):  # 加载主窗口  # 初始化窗口  pygame.display.init()  MainGame.window = pygame.display.set\_mode([SCREEN\_WIDTH,SCREEN\_HEIGHT])  # 初始化我方坦克  self.createMytank()  # 初始化敌方坦克，并将敌方坦克添加到列表中  self.createEnemyTank()  # 初始化墙壁  self.createWall()  # 设置窗口的标题  pygame.display.set\_caption("坦克大战1.0")  while True:  # 让坦克移动的速度慢一点  time.sleep(0.02)  # 对当前的窗口设置填充色  MainGame.window.fill(BG\_COLOR)  # 获取事件  self.getEvent()  # 调用绘制文字的方法  MainGame.window.blit(self.getTextSuface("敌方坦克剩余数量%d" % len(MainGame.enemyTankList)), (10, 10))  # 调用坦克显示的方法  # 判断我方坦克是否是否存活  if MainGame.my\_tank and MainGame.my\_tank.live:  MainGame.my\_tank.displayTank()  else:  # 删除我方坦克  del MainGame.my\_tank  MainGame.my\_tank = None  # 循环遍历敌方坦克列表，展示敌方坦克  self.blitEnemyTank()  # 循环遍历显示我方坦克的子弹  self.blitMyBullet()  # 循环遍历敌方子弹列表，展示敌方子弹  self.blitEnemyBullet()  # 循环遍历爆炸列表，展示爆炸效果  self.blitExplode()  # 循环遍历墙壁列表，展示墙壁  self.blitWall()  # 调用移动方法  # 如果坦克的开关是开启，才可以移动  if MainGame.my\_tank and MainGame.my\_tank.live:  if not MainGame.my\_tank.stop:  MainGame.my\_tank.move()  # 检测我方坦克是否与墙壁发生碰撞  MainGame.my\_tank.hitWall()  # 检测我方坦克是否与敌方坦克发生碰撞  MainGame.my\_tank.myTank\_hit\_enemyTank()  pygame.display.update()  # 循环遍历墙壁列表，展示墙壁  def blitWall(self):  for wall in MainGame.wallList:  # 调用墙壁的显示方法  if wall.live:  wall.displayWall()  else:  MainGame.wallList.remove(wall)  # 初始化墙壁  def createWall(self):  for i in range(6):  # 初始化墙壁  wall = Wall(i \* 130, 220)  # 将墙壁添加到列表中  MainGame.wallList.append(wall)  # 创建我方坦克的方法  def createMytank(self):  MainGame.my\_tank = MyTank(350, 300)  # 创建Music对象  music = Music('img/start.wav')  # 播放音乐  music.play()  # 循环遍历敌方坦克列表，展示敌方坦克  def blitEnemyTank(self):  for enemyTank in MainGame.enemyTankList:  if enemyTank.live:  enemyTank.displayTank()  enemyTank.randMove()  # 判断敌方坦克是否碰到墙  enemyTank.hitWall()  # 检测敌方坦克是否与我方坦克发生碰撞  if MainGame.my\_tank and MainGame.my\_tank.live:  enemyTank.enemyTank\_hit\_myTank()  # 敌方坦克发射子弹  enemyBullet = enemyTank.shot()  # 敌方子弹是否是None，如果不为None则添加到敌方子弹列表中  if enemyBullet:  # 将敌方子弹存储到敌方子弹列表中  MainGame.enemyBulletList.append(enemyBullet)  else: # 如果坦克已经死了，移除坦克  MainGame.enemyTankList.remove(enemyTank)  # 循环遍历显示我方坦克的子弹  def blitMyBullet(self):  for myBullet in MainGame.myBulletList:  # 判断子弹是否存在  if myBullet.live:  myBullet.displayBullet()  # 调用子弹的移动方法  myBullet.move()  # 调用检测我方子弹是否与敌方坦克发生碰撞  myBullet.myBullet\_hit\_enemyTank()  # 检测我方子弹是否与墙壁碰撞  myBullet.hitWall()  else:  MainGame.myBulletList.remove(myBullet)  # 循环遍历敌方子弹列表，展示敌方子弹  def blitEnemyBullet(self):  for enemyBullet in MainGame.enemyBulletList:  if enemyBullet.live: # 判断敌方子弹是否存活  enemyBullet.displayBullet()  enemyBullet.move()  # 调用敌方子弹与我方坦克碰撞的方法  enemyBullet.enemyBullet\_hit\_myTank()  # 检测敌方子弹是否与墙壁碰撞  enemyBullet.hitWall()  else:  MainGame.enemyBulletList.remove(enemyBullet)  # 初始化敌方坦克，并将敌方坦克添加到列表中  def createEnemyTank(self):  top = 100  # 循环生成敌方坦克  for i in range(MainGame.enemyTankCount):  left = random.randint(0, 600)  speed = random.randint(1, 4)  enemy = EnemyTank(left, top, speed)  MainGame.enemyTankList.append(enemy)  # 循环展示爆炸效果  def blitExplode(self):  for explode in MainGame.explodeList:  # 判断是否活着  if explode.live:  # 展示  explode.displayExplode()  else:  # 在爆炸列表中移除  MainGame.explodeList.remove(explode)  # 结束游戏  def endGame(self):  print('谢谢使用，欢迎再次使用')  exit()  # 左上角文字的绘制  def getTextSuface(self,text):  # 初始化字体模块  pygame.font.init()  # 查看所有的字体  # print(pygame.font.get\_fonts())  font = pygame.font.SysFont("stsong",18)  # 绘制文字信息  textSurface = font.render(text, True,TEXT\_COLOR)  return textSurface  # 获取事件  def getEvent(self):  # 获取所有事件  eventList = pygame.event.get()  # 遍历事件  for event in eventList:  # 判断按下的键是关闭还是键盘按下  # 如果按的是退出，关闭窗口  if event.type == pygame.QUIT:  self.endGame()  # 如果是键盘的按下  if event.type == pygame.KEYDOWN:  if MainGame.my\_tank and MainGame.my\_tank.live:  # 判断按下的是上、下、左、右  if event.key == pygame.K\_LEFT:  # 切换方向  MainGame.my\_tank.direction = "L"  # 修改坦克是否移动的开关  MainGame.my\_tank.stop = False  # MainGame.my\_tank.move()  print('按下左键，坦克向左移动')  elif event.key == pygame.K\_RIGHT:  # 切换方向  MainGame.my\_tank.direction = "R"  # 修改坦克是否移动的开关  MainGame.my\_tank.stop = False  # MainGame.my\_tank.move()  print('按下右键，坦克向右移动')  elif event.key == pygame.K\_UP:  # 切换方向  MainGame.my\_tank.direction = "U"  # 修改坦克是否移动的开关  MainGame.my\_tank.stop = False  # MainGame.my\_tank.move()  print('按下上键，坦克向上移动')  elif event.key == pygame.K\_DOWN:  # 切换方向  MainGame.my\_tank.direction = "D"  # 修改坦克是否移动的开关  MainGame.my\_tank.stop = False  # MainGame.my\_tank.move()  print('按下下键，坦克向下移动')  elif event.key == pygame.K\_SPACE:  print("坦克发射子弹")  if len(MainGame.myBulletList) <= 3: # 最多可以存在四颗子弹  # 创建我方坦克发射的子弹  myBullet = Bullet(MainGame.my\_tank)  # 将发射的子弹添加到列表  MainGame.myBulletList.append(myBullet)  # 我方坦克发射子弹添加音效  music = Music('img/hit.wav')  music.play()  # 当坦克不存在活着死亡时  else: # 按下R键，坦克重生  if event.key == pygame.K\_r:  self.createMytank()  # 松开方向键，坦克停止移动  if event.type == pygame.KEYUP:  # 判断松开的键是上、下、左、右时候才停止坦克移动  if event.key == pygame.K\_UP or event.key == pygame.K\_DOWN or event.key == pygame.K\_LEFT or event.key == pygame.K\_RIGHT:  if MainGame.my\_tank and MainGame.my\_tank.live:  if (MainGame.my\_tank.stop == False): # 如果坦克正在移动 并且方向始终如一的时候 才停止  if (event.key == pygame.K\_UP and MainGame.my\_tank.direction == "U"):  MainGame.my\_tank.stop = True  elif (event.key == pygame.K\_DOWN and MainGame.my\_tank.direction == "D"):  MainGame.my\_tank.stop = True  elif (event.key == pygame.K\_LEFT and MainGame.my\_tank.direction == "L"):  MainGame.my\_tank.stop = True  elif (event.key == pygame.K\_RIGHT and MainGame.my\_tank.direction == "R"):  MainGame.my\_tank.stop = True |

1. **评价**

程序的可读性：

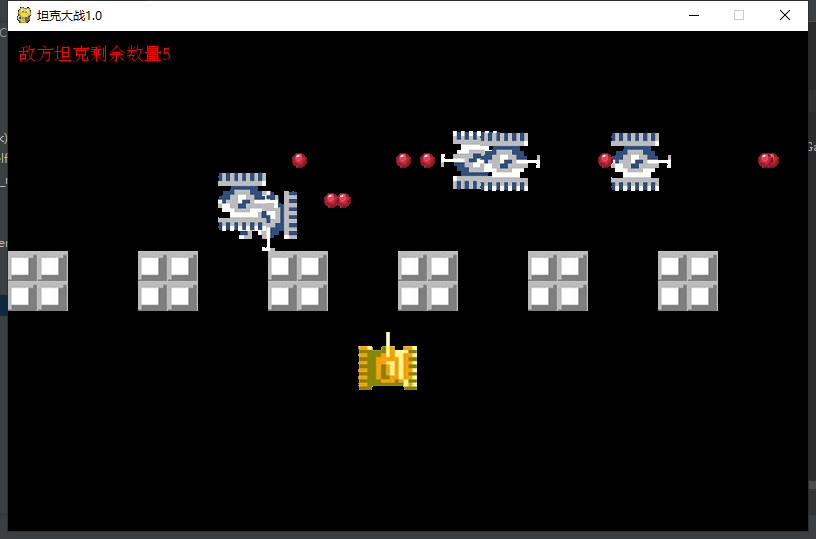
阅读上面代码片段，就可以看出，该程序的方法，变量命名，还有方法的注释，变量的注释，都详细的体现在上面的片段中，目的就是增加程序的可读性，便于以后的修改与升级，避免别人阅读代码一头雾水，摸不着头脑。

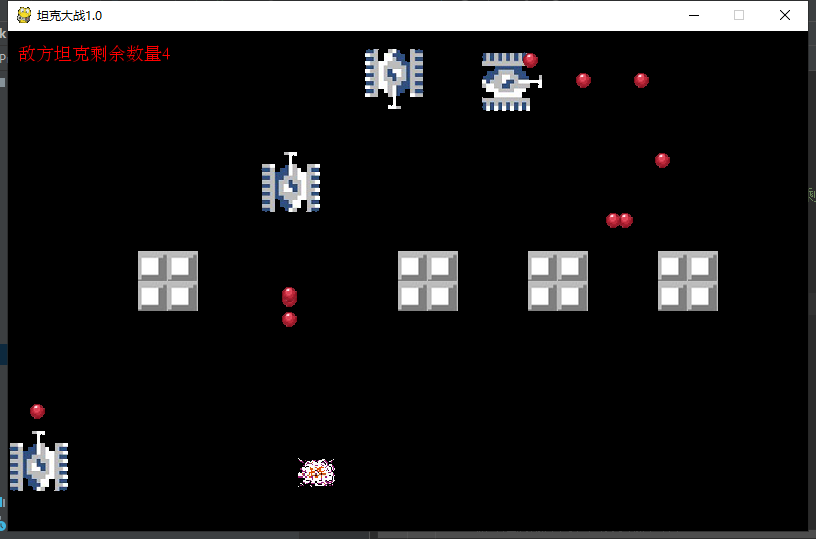


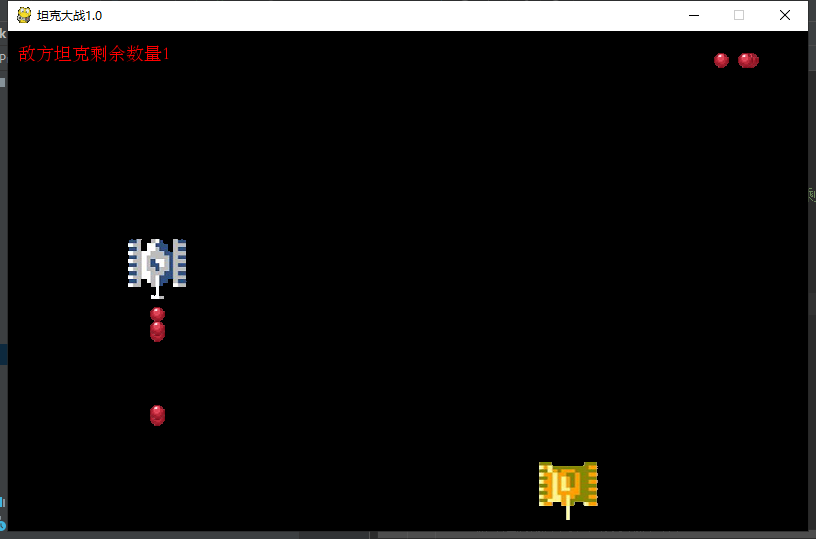
由图可知，程序运行时，主动调用的sleep函数占据了程序运行的77%的时间，可知程序的对硬件的要求与对内存交互时间都很少，显示地方的坦克剩余量的函数，也占据了程序运行时间的10%，表明程序的功能性的方法，实现该游戏的核心功能等方法占据了很少一部分资源。

1. **结论**

**运行截图：**







**总结：**

程序是一步一步进行的，显示编辑窗体，然后在将需要的部分一个一个的加入到窗体中，先增加了我方坦克，然后在增加敌方坦克，然后坦克可以发射子弹，在这期间，每进行一步，都需要对之前的代码进行完善与修改，在修改的过程中，我深刻的体会到在编码前进行详细的概要设计，将程序的接口，方法等都事先定义好有多么的重要，如果没有事先定义好每一个模块的方法，成员变量等。会让你在一步一步完善功能的过程中，将代码变得难以阅读，使程序的结构完整性受到严重的破坏，进行良好的程序架构设计，会让你在一步一步的代码不断完善的过程中，可以很方便的在对应的位置添加变量，添加需要的代码，而不是哪里出现问题，哪里需要添加代码，就在哪里添加，到最后完成项目的时候，使自己都难以阅读自己的代码，如果程序出现问题，将更加难以修改。并且，书写注释也是一个很好的编程习惯，每一行代码做了什么，每一个变量用来记录什么，都显示的一清二楚，增加了程序的可读性。