# 飞机大战项目说明书

> 数据科学与大数据技术 车宇庚 大数据202 2029730202 指导教师: 王继强

## 项目结构

把游戏分为

- 游戏面板模块 game_hud.py 封装指示器
- 游戏元素 game_items.py 封装英雄 敌人 道具 等并定义全局变量
- 音乐 game_music.py 封装游戏播放器类
- 游戏主模块部分 game.py 负责启动游戏封装Game类

### 游戏面板模块

```
class HudPanel:
# 内置对象变量:精灵之间的间距 颜色 各关卡的分数阈值和各种函数
    margin = 10  # 精灵之间的间距
    white = (255, 255, 255)  # 白色
    gray = (64, 64, 64)  # 灰色

    reward_score = 100000  # 关卡奖励分数
    level2_score = 10000  # 关卡级别2的预设分值
    level3_score = 50000  # 关卡级别3的预设分值

    record_filename = "record.txt"
```

```
    def __init__(self, display_group):
        """构造方法:param display_group:面板中的精灵要被添加到的显示精灵组"""
        # 这里会初始化游戏属性 如生命数 关卡级别 最好成绩 当前成绩
        #也会创建炸弹精灵 状态精灵生命计数精灵
        #创建标签精灵 如分数标签 炸弹标签生命计数标签等
        self.score = 0  # 游戏得分
        self.lives_count = 3  # 生命数
        self.level = 1  # 关卡级别
        self.best_score = 0  # 最好成绩
```

```
    def show_lives(self):
        """显示生命计数"""
        # 设置生命计数标签文字
        # 设置生命计数标签位置
        # 调整生命计数精灵位置
```

```
    def increase_score(self, enemy_score):
        """增加游戏得分:param enemy_score:摧毁敌机的分值
        :return: 增加enemy_score后，关卡级别是否提升
        """
        # 游戏加分
        # 判断是否奖励生命
        # 最好成绩保存
        # 关卡级别改变
        # 修改得分
```

```
    def save_best_score(self):
        """将最好成绩写入record.txt"""
```

```
    def load_best_score(self):
        """从record.txt加载最好成绩"""
```

```
    def panel_pause(self, is_game_over, display_group):
        """面板暂停
        :param is_game_over:是否因为游戏结束需要停止
        :param display_group: 显示精灵组
        """
        # 判断是否已经添加了精灵
        # 根据是否结束游戏决定要显示的文字:"Game Over!";"Game Paused!"......
        # 设置暂停标签文字和位置 添加到精灵组 切换精灵的状态
```

```
    def panel_resume(self, display_group):
        """面板恢复
        :param display_group:显示精灵组
        """
        # 从精灵组移除3个标签精灵 切换精灵状态
```

```
    def reset_panel(self):
        #重置面板
```

## 游戏元素

```
SCREEN_RECT = pg.Rect(0, 0, 480, 700)          # 游戏主窗口矩形区域
FRAME_INTERVAL = 10                                 # 逐帧动画间隔帧数

HERO_BOMB_COUNT = 3                                 # 英雄默认炸弹数量
# 英雄默认初始位置
HERO_DEFAULT_MID_BOTTOM = (SCREEN_RECT.centerx,
                           SCREEN_RECT.bottom - 90)

HERO_DEAD_EVENT = pg.USEREVENT                  # 英雄牺牲事件
HERO_POWER_OFF_EVENT = pg.USEREVENT + 1         # 取消英雄无敌事件
HERO_FIRE_EVENT = pg.USEREVENT + 2              # 英雄发射子弹事件

THROW_SUPPLY_EVENT = pg.USEREVENT + 3           # 投放道具事件
BULLET_ENHANCED_OFF_EVENT = pg.USEREVENT + 4  # 关闭子弹增强事件
```

```
class GameSprite(pg.sprite.Sprite):
    #继承了pygame的sprite
    #重写了init()和update()
    self.image = pg.image.load(self.res_path + image_name) #图片
    self.rect = self.image.get_rect()# 位置
    self.speed = speed# 速度
    self.mask = pg.mask.from_surface(self.image)# pygame.mask - 用于快速的像素完美碰撞检测。
```

```
class Label(pg.sprite.Sprite):
    #继承了GameSprite 成为了标签类 设置字体文字然后主要用于 上述控件元素部分的标签类的实例化 如分数标签 炸弹标签生命计数标签等
    def set_text(self, text):
        #传入文字设置显示的内容
```

```
class Background(GameSprite):
    #继承了GameSprite类 成为了背景类游戏中的背景
    def update(self, *args):
        #实现背景的动态效果
```

```
class StatusButton(GameSprite):
    #继承了GameSprite类 成为按键精灵 用于控制游戏页面中的按钮变化
    def switch_status(self, is_pause):
        #变更按钮的样式(暂停或继续)
```

```
class Plane(GameSprite):
    #继承了GameSprite类 成为飞机类 定义了飞机各种参数
        def __init__(self, hp, speed, value, wav_name,
                     normal_names, hurt_name, destroy_names, *groups):
        """
        :param normal_name: 记录正常飞行状态的图像名称列表
        :param groups: 要添加到的精灵组
        """
        #定义了飞机的血量 正常飞行时的图片和 受伤的照片和被摧毁时的照片和各种参数 刻画了一个飞机的模型
        def reset_plane(self):
            #重置飞机的参数 恢复血量 状态改为没有被摧毁且正常的状态 显示图标变为正常的
        def update(self, *args):
        #用于飞机被攻击或复活后状态的变化 血量的改变 图标的改变和判断是否坠毁
```

```
class Enemy(Plane):
    #继承了 Plane类建立的敌机类用于表示游戏中对面敌人
    def __init__(self, kind, max_speed, *groups):
        #用与建立不同类型的敌机 :不同的速度不同的 血量不同的图标 等
    def reset_plane(self):
        #重写了Plane的reset_plane
        #用于重置飞机位置和参数  对于飞机的移动和复活的控制
    def update(self, *args):
        #用于飞机血量的改变和根据速度位置的改变
```

```
class Hero(Plane):
    #继承Plane的英雄飞机类 是玩家操作的
```

```
        def __init__(self, *groups):
        """
        :param groups: 要添加到的精灵组
        """
        #定义了飞机的血量 正常飞行时的图片和 受伤的照片和被摧毁时的照片和各种参数 刻画了一个飞机的模型
        #设置是否无敌 炸弹数量 子弹类型 位置等
        def reset_plane(self):
            #重置飞机的参数 恢复血量 状态改为没有被摧毁且正常的状态 显示图标变为正常的 和重新设置是否无敌位置 炸弹类型 子弹类型
        def update(self, *args):
        """
        :param args: 0 更新图像标记 1 水平移动基数 2 垂直移动基数
        """
        # 调用父类方法更新飞机图像 args要解包 如果没有传递方向参数或者飞机坠毁那么直接返回 调整水平移动距离 限定在窗口内移动
    def blowup(self, enemies_group):
        """
        :param enemies_group: 敌机精灵组
        :return: 累计得分
        """
        # 如过没有足够的炸弹或者敌机被摧毁那么直接返回
        # 遍历敌机精灵组，将游戏窗口内的敌机引爆
    def fire(self, display_group):
        """
        :param display_group: 要添加的显示精灵组
        """
        #发出子弹 初始化子弹类的参数 和位置
```

```
class Bullet(GameSprite):
    #继承GameSrite
    """子弹类"""
    def __init__(self, kind, *groups):
        """
        :param kind: 子弹类型
        :param groups: 要添加到的精灵组
        #设置子弹图标和类型 速度等

    def update(self, *args):
    #如果出界就自动销毁
```

```
class Supply(GameSprite):
    #继承GameSrite 道具类 大招等东西
        def __init__(self, kind, *groups):
        """
        :param kind: 道具类型
        :param groups: 要添加到的精灵组
        """
        #设置道具的图标 音效和位置
        def throw_supply(self):
            #设置道具的位置
        def update(self, *args):
            #更新位置 和判断是否出界
```

## 音乐

```
class MusicPlayer:
    def __init__(self, music_file):
        #设置背景音乐
        #设置音乐的列表
        #播放器初始化
    @staticmethod
    def play_music(time):
        #播放
    @staticmethod
    def pause_music(is_pause):
        #暂停
    def play_sound(self, wav_name):
    """
    :param wav_name:音效文件名
    :return:
    """
    #设置播放的文件
```

## 游戏主模块

```
class Game(object):
    #游戏主类 整个各个模块 提供游戏
        self.main_window = pg.display.set_mode(SCREEN_RECT.size)#游戏窗口
        pg.display.set_caption("Plane War")# 游戏名
        self.is_game_over = True# 是否结束
        self.is_pause = False# 时候暂停
        self.all_group = pg.sprite.Group()# 精灵组
        self.enemies_group = pg.sprite.Group()# 敌人
        self.supplies_group = pg.sprite.Group()#道具组
        self.all_group.add(Background(False), Background(True))# 背景
        self.hud_panel = HudPanel(self.all_group)#控制面板
        self.create_enemies()# 创建敌机
        self.hero = Hero(self.all_group)# 创建英雄
        self.hud_panel.show_bomb(self.hero.bomb_count)#显示炸弹
        self.create_supplies()#创建道具
        self.player = MusicPlayer("game_music.ogg")#实例化背景音乐
        MusicPlayer.play_music(-1)#循环播放背景音乐
    def __init__(self):
        #初始化参数 窗口大小 暂停 精灵组 敌人 道具 面板 英雄飞机类 音乐播放器类
    def create_supplies(self):
        #创建道具 游戏开始后每三十秒随机投放炸弹补给或子弹道具
    def create_enemies(self):
        # 敌机精灵组中的精灵数量
        # 要添加到的精灵组
        # 判断游戏级别及已有的敌机数量
    def reset_game(self):
        #初始化游戏在新游戏开始后将游戏属性恢复到初始值
    def event_handler(self):
        """
        :return: 如果监听到退出事件，返回 True，否则返回 False
        """
        #事件监听 监听并处理游戏循环执行时发生的事件,避免循环代码过长
    def start(self):
```

```
        #开始游戏,创建时钟对象并开启游戏循环 在循环中监听事件 更新精灵位置 绘制精灵 更新显示 设置刷新频率
    def check_collide(self):
        #碰撞检测 监听并处理每一次游戏循环执行时时候发生精灵之间的碰撞 例如 子弹击中飞机 英雄拾取道具 敌机撞击英雄
```

## Main

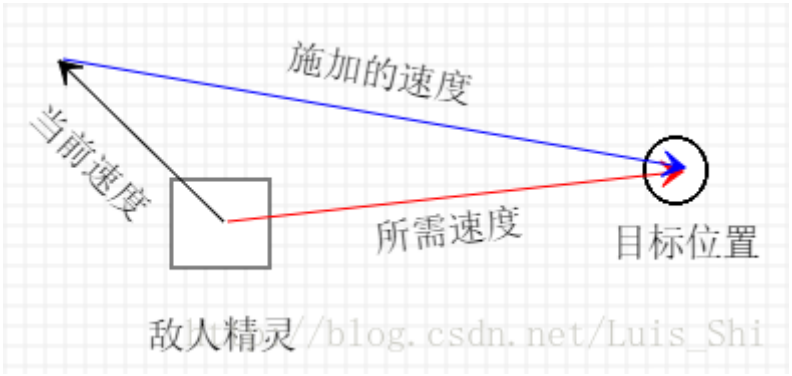
```
if __name__ == '__main__':
    pg.init() #初始化
    Game().start() #游戏入口
    pg.quit() #是一个取消初始化所有pygame模块的函数。此函数应在应用程序结束时调用:
```

# 个人感想

## 学到的

- 怎么实现敌方精灵的自动追踪?

  在查阅了相关资料后有了解决办法—— 转向力 = 所需速度 - 当前速度

  在上述公式中，我们已知精灵的当前速度和目标位置只需要计算施加的速度即可，我们可以这样设计敌人精灵，

- Pygame 中处理图形遮罩的模块。
  - **pygame.mask.from_surface()**

    从指定 Surface 对象中返回一个 Mask。
  - **pygame.mask.from_threshold()**

    从给定阈值的 Surface 对象中创建一个 Mask。
  - 通常一个游戏中会有很多角色出现，而这些角色之间的“碰撞”在所难免，例如炮弹是否击中了飞机等。碰撞检测在绝大多数游戏中都是一个必须处理的至关重要的问题。pygame提供了多种碰撞的检测方法，包括矩形碰撞检测、圆形碰撞检测和使用mask的精准碰撞检测。
- 游戏大致来讲是由动画和人机交互的体验两部分构成，其中动画则是由一系列连续静止的图片，经过一定频率的刷新构成的，这个频率被称为 FPS，如果频率值越大则画面越流畅；如果频率值越小则画面会出现卡顿的感，在游戏过程中一般人能接受的最低 FPS 约为 30Hz，如果想要画面流畅则 FPS 要大于 60 Hz。

  > FPS 越高，细节越好，体验也越好，但是文件容量也越高

- 动画保证了玩家的视觉体验，而人机交互则是操作上的体验。通过移动和点击鼠标、按下键盘上的技能键，或是滑动手机屏幕等操作来实现人机交互，这些与游戏程序交互的操作被称为事件（Event）。

```
# 循环获取事件，监听事件状态，使用get()获取事件
for event in pygame.event.get():
    # 判断事件类型，用户是否点了"X"关闭按钮
    # pygame.QUIT 指点击右上角窗口的"X"号
    if event.type == pygame.QUIT:
        #点击后，卸载所有pygame模块
        pygame.quit()
```

- 游戏循环
  - 当打我们游戏时可能会触发游戏中的各种事件，比如鼠标事件、键盘按键事件、摄像拍照事件等等，因此游戏程序需要一直循环监听玩家的操作，只有当用户点击了游戏“关闭”按钮时，监听才会结束。如果想要达到“循环监听”目的，此时就需要设置一个游戏循环（Game Loop）也称为游戏的主循环，这样才能保证人机交互的体验感。代码示例如下：

```
#游戏主循环(游戏循环)
while True:
    # 循环获取事件，监听事件
    for event in pygame.event.get():
        # 判断用户是否点了关闭按钮
        if event.type == pygame.QUIT:
            # 当用户关闭游戏窗口时执行以下操作
            # 这里必须调用quit()方法，退出游戏
            pygame.quit()
            #终止系统
            sys.exit()
    #更新并绘制屏幕内容
    pygame.display.flip()
```

- Pygame 专门提供了一个处理精灵的模块，也就是 sprite（pygame.sprite）模块。通常情况下，我们使用该模块的基类 Sprite 来创建一个子类，从而达到处理精灵的目的，该子类提供了操作精灵的常用属性和方法，如下所示：

| 属性&方法 | 说明 |
| --- | --- |
| self.image | 加载要显示的精灵图片，控制图片大小和填充色 |
| self.rect | 精灵图片显示在哪个位置 |
| Sprite.update() | 刷新精灵图，使其相应效果生效 |
| Sprite.add() | 添加精灵图到精灵组中（groups） |
| Sprite.remove() | 从精灵组中删除选中的精灵图 |
| Sprite.kill() | 删除精灵组中全部的精灵 |
| Sprite.alive() | 判断某个精灵是否属于精灵组 |

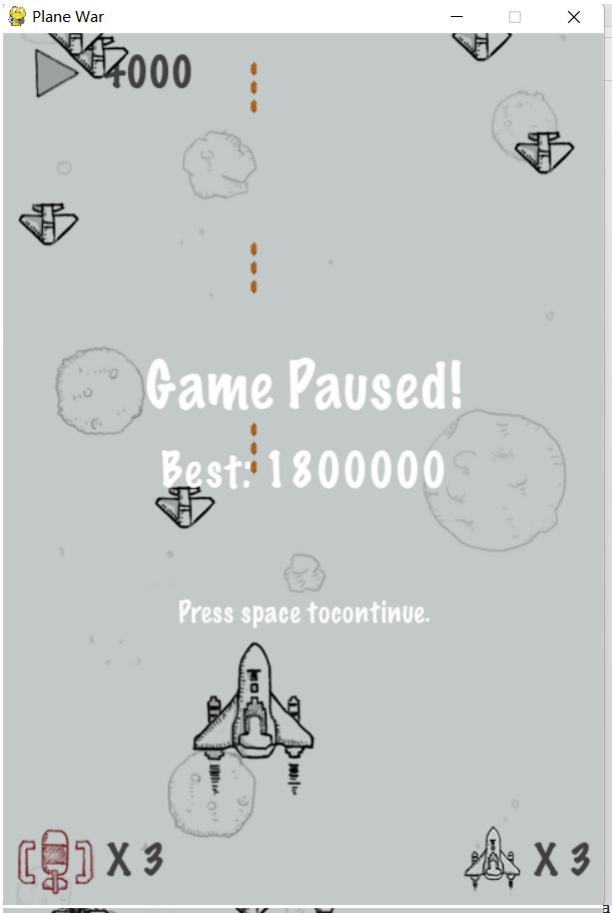
- 当游戏中有大量的精灵时，操作它们将变得复杂，此时通过构建精灵容器（group 类）也就是精灵组来统一管理这些精灵。构建方法如下：

```
# 创建精灵组
group = pygame.sprite.Group()
# 向组内添加一个精灵
group.add(sprite_one)
```

于此同时`pygame.sprite`模块也提供了多种检测精灵是否碰撞的方法，如下所示：

| | |
| --- | --- |
| pygame.sprite.collide_circle() | 两个精灵之间的圆形检测，即圆形区域是否有交汇，返回一个布尔值。 |
| pygame.sprite.collide_mask() | 两个精灵之间的像素蒙版检测，更为精准的一种检测方式。 |
| pygame.sprite.spritecollide() | 精灵和精灵组之间的矩形碰撞检测，一个组内的所有精灵会逐一地对另外一个单个精灵进行碰撞检测，返回值是一个列表，包含了发生碰撞的所有精灵。 |
| pygame.sprite.spritecollideany() | 精灵和精灵组之间的矩形碰撞检测，上述函数的变体，当发生碰撞时，返回组内的一个精灵，无碰撞发生时，返回 None。 |
| pygame.sprite.groupcollide() | 检测在两个组之间发生碰撞的所有精灵，它返回值是一个字典，将第一组中发生碰撞的精灵作为键，第二个组中发生碰撞的精灵作为值。 |

# 展示:





# 最后

**python真简单**

人生苦短我用python