OOP式开发游戏

通过前面的游戏设计, 基本明白, 飞机大战游戏就分**两大文件**:

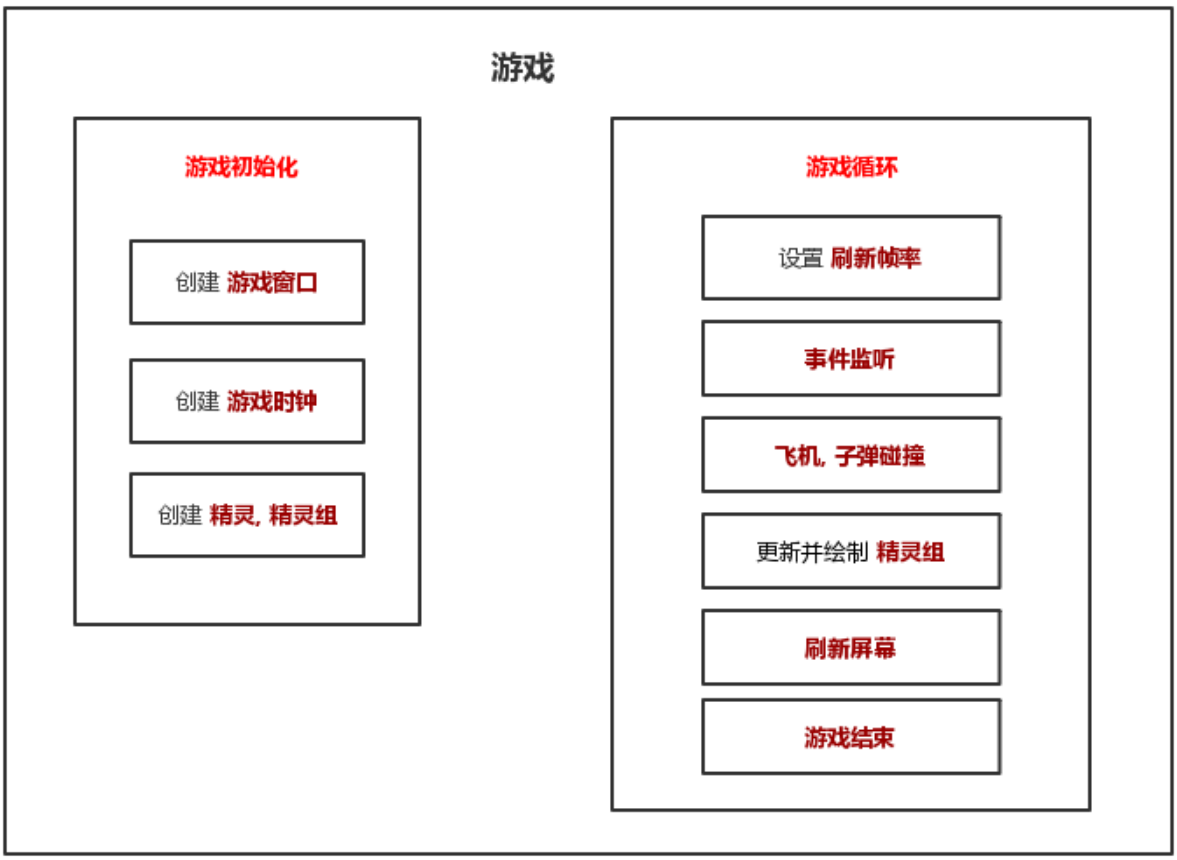
- 游戏**主程序** plane.py
- 角色**精灵** plane_role.py

主程序

主程序文件: plane.py

主程序主要就**负责**两件事情:

- 游戏**初始化**
- **游戏循环**



游戏初始化

```
import pygame

# 窗口矩形
SCREEN_RECT = pygame.Rect(0,0,480,700)

class Plane():

    # 游戏初始化
    def __init__(self):
        print('游戏初始化')

        # 1. 创建窗口
        self.screen = pygame.display.set_mode((SCREEN_RECT.width, SCREEN_RECT.height))
)

        # 2. 创建时钟
        self.clock = pygame.time.Clock()

        # 3. 创建各种飞机图像
        pass

        # 4. 创建精灵，精灵组
        pass

    # 游戏循环
    def start_game(self):
        print('游戏开始')


game = Plane()
game.start_game()
```

要点分析:

SCREEN_RECT = pygame.Rect(0,0,480,700)

在python中, 若变量以 **纯大写**形式出现, 是想以**"常量"** 的身份出现.

在很多语言中, **常量**一般存储**固定值**, 例如: 数学中的 π, 自然底数e, 等经常不会变的数值

常量优势

- 见名知其意
- 若调整所有的**固定值**, 只需要修改常量值即可

注意点

- 常量名一般都是 **纯大写**
- 定义后, 少改动

游戏开始

框架步骤

```
def start_game(self):
    print('游戏开始')

    while True:
        # 1. 刷新帧率
        self.clock.tick(FRAME_NUM)

        # 2. 事件监听
        self.__event_monitor()

        # 3. 飞机, 子弹碰撞
        self.__collision()

        # 4. 更新/绘制精灵组
        self.__update_sprite()

        # 5. 刷新屏幕
        pygame.display.update()

# 事件监听
def __event_monitor(self):
    pass

# 碰撞
def __collision(self):
    pass

# 更新精灵组
def __update_sprite(self):
    pass

# 游戏结束
@staticmethod
def __game_over():
    print('游戏结束')
    pygame.quit()
    exit()
```

要点分析

在 **start_game** 中, 通过 **无限循环** 来保持窗口显示

在 **无限循环** 中不断的 **刷新帧数**, **监听事件**, **飞机子弹碰撞**, **更新绘制精灵组**, **刷新屏幕**

将 **无限循环** 的功能全部独立出来, 方便维护

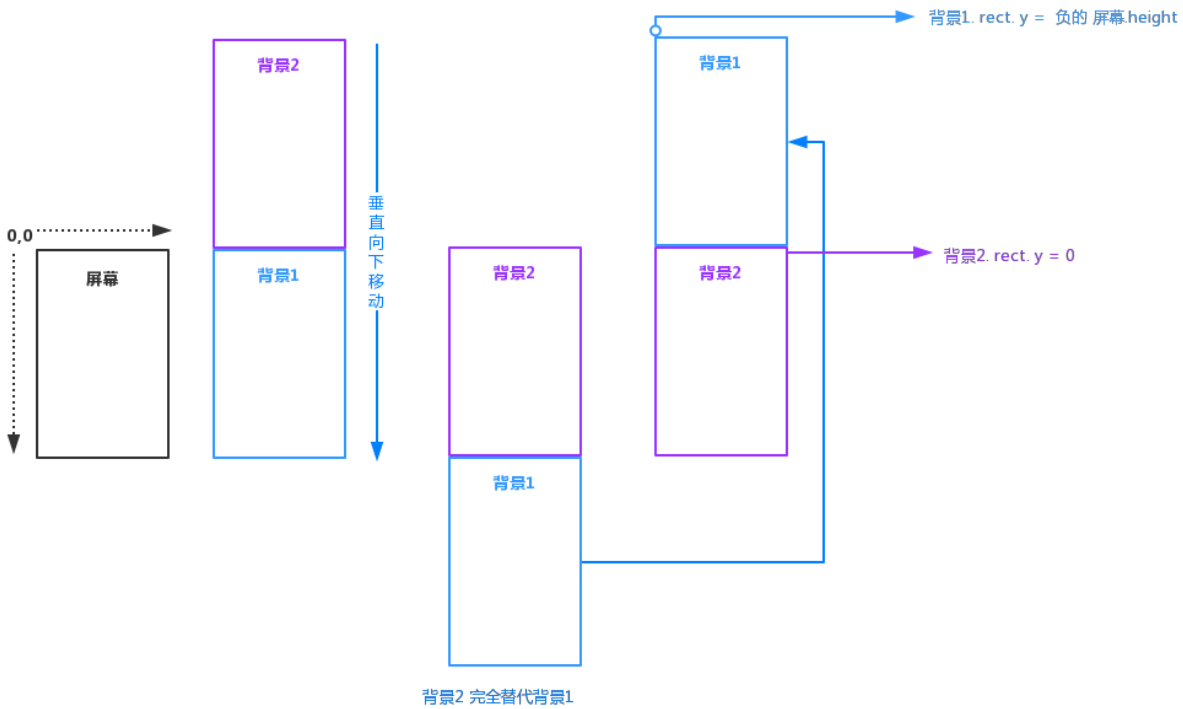
刷新帧数中, 可以直接数字. 但由于帧数一般不会变, 所以设置为 **常量** 更合适

游戏结束, 不适合直接写在 **无限循环** 中, 不是每次循环都需要结束游戏的. 结束游戏应该是在 **特定场景** 下才会调用. 例如: 英雄牺牲, 窗口关闭等

游戏背景

交替滚动背景

设计原理:



创建两个背景

1. 背景1 与 屏幕完全吻合
2. 背景2 在背景1正上方, 紧贴着

交替滚动

1. 两张背景同时向下滚动
2. 当背景1 完全脱离屏幕时, 立马移动到背景2 的正上方, 紧贴着
3. 此时, 背景2 与 屏幕完全吻合, 两张背景同时向下滚动
4. 一旦背景2 完全脱离屏幕时, 重复第二个步骤. 反复行驶, 就形成了无限背景滚动的样子了

plane.py

```
创建精灵
def __create_sprite(self):
    bg1 = Background('./image/background.png')
    bg2 = Background('./image/background.png')
    bg2.rect.y = -SCREEN_RECT.height

更新精灵组
def __update_sprite(self):
    self.bg.update()
    self.bg.draw(self.screen)
```

plane_role.py

```
SCREEN_RECT = pygame.Rect(0,0,480,700) # 窗口矩形
FRAME_NUM = 60 # 帧数

# 背景精灵
class Background(pygame.sprite.Sprite):
    def __init__(self, image, speed=1):
        # 调用精灵类的初始化
        super().__init__()

        # 获取图像
        self.image = pygame.image.load(image)
        # 获取图像矩形
        self.rect = self.image.get_rect()
        # 获取速度
        self.speed = speed



    def update(self):
        # 1. 背景 向下移动
        self.rect.y += self.speed

        # 2. 判断是否移除屏幕，若移出，则将图像移至 屏幕上方
        if self.rect.y >= SCREEN_RECT.height:
            self.rect.y = -SCREEN_RECT.height
```

简化代码

在主程序中, 背景都是一样的, 但每次都需要在输入一次背景图片地址. 对于类似 固定重复性代码, 可做以下简化

plane.py

```
创建精灵
def __create_sprite(self):
    # 简化写法（需调整精灵）
    bg1 = Background()
    bg2 = Background(is_repeat=True)

    self.bg = pygame.sprite.Group(bg1, bg2)
```

plane_role.py

```
SCREEN_RECT = pygame.Rect(0,0,480,700) # 窗口矩形
FRAME_NUM = 60 # 帧数

# 背景精灵
class Background(pygame.sprite.Sprite):
    def __init__(self, image='./image/background.png', speed=1, is_repeat=False):
        # 调用精灵类的初始化
        super().__init__()

        # 获取图像
        self.image = pygame.image.load(image)
        # 获取图像矩形
        self.rect = self.image.get_rect()
        # 获取速度
        self.speed = speed

        # 判断是否为第二张图像
        if is_repeat == True:
            self.rect.y = -SCREEN_RECT.height


    def update(self):
        # 1. 背景 向下移动
        self.rect.y += self.speed

        # 2. 判断是否移除屏幕，若移出，则将图像移至 屏幕上方
        if self.rect.y >= SCREEN_RECT.height:
            self.rect.y = -SCREEN_RECT.height
```

敌方飞机

- 定时器
- 敌方精灵
- 显示敌机

定时器

定义: 每隔一段时间, 就会执行一些功能

定时器步骤:

1. 设置 **自定义事件**
2. 设置**定时 触发**自定义事件
3. **监听** 自定义事件

1. 自定义事件

除了pygame自带的 鼠标, 键盘等事件外, 用户也可以自定义事件.

通过 USEREVENT 来设置自定义事件

```
事件名 = pygame.USEREVENT
```

USEREVENT 会生成**事件id**, 与事件名进行绑定

此处的USEREVENT 生成的id是固定值, 想要再**自定义新的事件**时, 需要 **USEREVENT + 1**, 有更多的再加1, 用以**区分**. 以此类推

2. 定时 触发

每间隔一段时间, 就会执行一些事件. 通过set_time() 来设置

```
pygame.time.set_time(事件id, 毫秒)
```

3. 监听 自定义事件

在 pygame.event.get() 循环中, 判断当前事件 是否为 自定义事件

```
for event in pygame.event.get():
    if event.type == ENEMY_EVENT:
        每秒需要做的事情
```

敌方精灵

敌方精灵主要负责以下事情:

- 每隔1s 出现一架敌机
- 每架敌机 都向下飞行
- 飞行速度各不相同
- 水平出现位置各不相同
- 敌机飞出屏幕后, 需要删除敌机

```
# 敌机精灵
class Enemy(pygame.sprite.Sprite):
    def __init__(self, image):
        # 调用精灵类的初始化
        super().__init__()

        # 获取图像
        self.image = image
        # 获取图像矩形
        self.rect = self.image.get_rect()
        # 随机获取速度
        self.speed = random.randint(1,3)

        # 设置敌机 水平初始位置
        self.rect.x = random.randint(0, SCREEN_RECT.width-self.rect.width)

    def update(self):
        # 1. 背景 向下移动
        self.rect.y += self.speed

        # 2. 判断是否移除屏幕，若移出，将当前精灵 从精灵组中删除
        if self.rect.y >= SCREEN_RECT.height:
            self.kill()
```

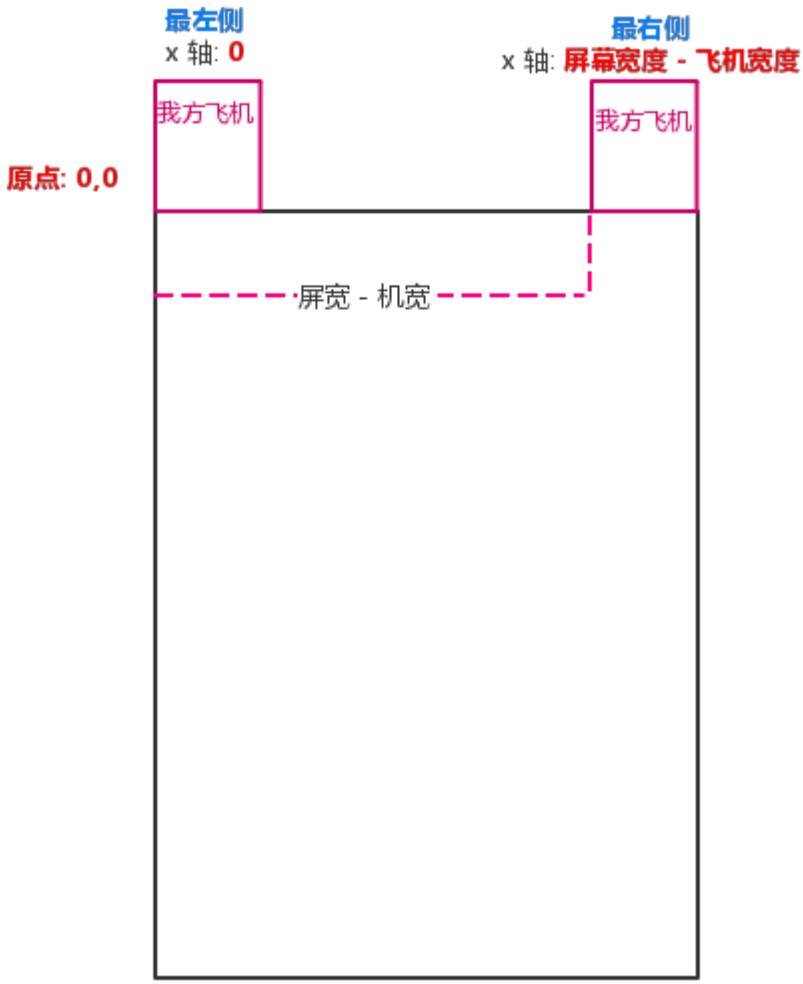
要点分析

1. 随机获取速度

通过 random 模块, 随机区间数字, 代表每次飞行的距离, 距离不同, 则代表 飞行速度不同

2. 敌机出现的 初始位置

敌机每次出现的位置, 应该是随机的, 但也不能超过屏幕宽度



3. 销毁飞出的敌机

敌机飞出屏幕, 理论上会一直向下飞. 内存一直被占用, 实际上已经没有任何意义了(都已经飞出屏幕, 又不能打)

解决方案:

通过 **精灵.kill()** 从精灵组中删除自己

显示敌机

- 抠图敌机图片
- 设置敌机事件 定时器
- 创建敌机精灵组
- 定时添加精灵
- 更新并显示敌机精灵

抠图敌机图片

```
def __init__(self):
    # 大图
    self.plane_img = pygame.image.load('./image/shoot.png')

    # 小敌机
    self.enemy1_rect = pygame.Rect(534, 612, 57, 43)
    self.enemy1_img = self.plane_img.subsurface(self.enemy1_rect)
```

设置敌机事件 定时器

```
精灵文件，常量
ENEMY_EVENT = pygame.USEREVENT

主程序
def __init__(self):
    pygame.time.set_timer(ENEMY_EVENT, 1000)
```

创建敌机精灵组

```
def __create_sprite(self):
    self.enemy_group = pygame.sprite.Group()
```

注意

按照往常习惯, 是要先**创建**一个**精灵**, 在送进**精灵组**.

但是, 这里敌机需要很多, 不止一架, 且出场时间, 也不一致.

所以, 此处只是先 准备好 精灵组, 后期, 通过**定时器**, 按时往 精灵组中 **添加精灵**

定时添加精灵

```
def __event_monitor(self):
    if event.type == ENEMY_EVENT:
        enemy1 = Enemy(self.enemy1_img)
        self.enemy_group.add(enemy1)
```

注意

只有在 **敌机事件** 中, 才会创建 **敌机精灵**

创建后, 再将 **敌机精灵** 添加进 **敌机精灵组**, 通过 **精灵组.add(精灵)** 执行

更新并显示敌机精灵

```
def __update_sprite(self):
    self.enemy.update()
    self.enemy.draw(self.screen)
```

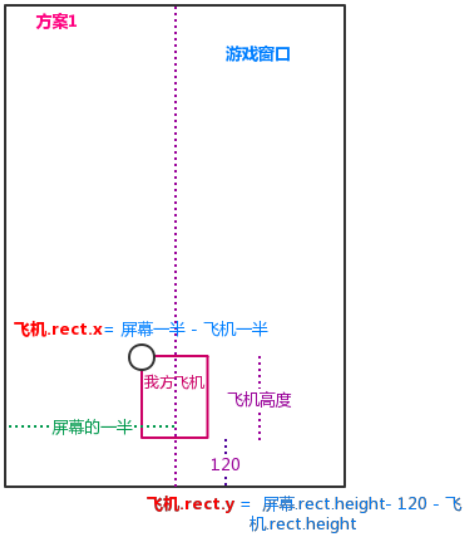
英雄飞机

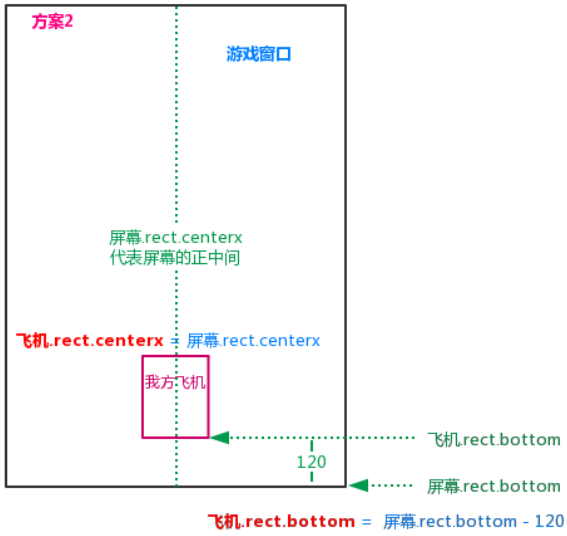
- 英雄精灵
- 显示英雄
- 移动英雄
- 边界控制

英雄精灵

- 设置英雄的**初始位置**
- 移动英雄
- 发射子弹

```
class Hero(pygame.sprite.Sprite):
    '''我方英雄飞机 精灵'''

    def __init__(self, image):
        super().__init__()

        # 获取图像 surface
        self.image = image
        # 获取图像矩形
        self.rect = self.image.get_rect()

        # 获取英雄飞机的初始位置
        # 方案1
        self.rect.x = SCREEN_RECT.width/2 - self.rect.width/2
        self.rect.y = SCREEN_RECT.height - self.rect.height - 120

        # 方案2
        # self.rect.centerx = SCREEN_RECT.centerx
        # self.rect.bottom = SCREEN_RECT.bottom - 120

    # 移动飞机
    def update(self):
        pass

    # 发射子弹
    def fire(self):
        pass
```





要点分析

方案1:

飞机x 坐标 = 屏幕的一半 - 飞机的一半

飞机y 坐标 = 屏幕高度 - 飞机高度 - 离屏幕底部的距离

方案2:

飞机 正中间的位置 = 屏幕正中间的位置

飞机的底部 = 屏幕的底部 - 离屏幕底部的距离

显示英雄

- 抠图英雄图片
- 创建英雄精灵组
- 更新并显示英雄精灵

抠图英雄图片

```
def __init__(self):
    # 大图
    self.plane_img = pygame.image.load('./image/shoot.png')

    # 英雄飞机
    self.hero_rect = pygame.Rect(0, 99, 102, 126)
    self.hero_img = self.plane_img.subsurface(self.hero_rect)
```

注意: 如果英雄图片是单独的, 则不需要抠图, 直接创建英雄精灵

创建英雄精灵组

```
def __create_sprite(self):
    self.hero = Hero(self.hero_img)
    self.hero_group = pygame.sprite.Group(self.hero)
```

注意: 后期英雄需要 调用**非update方法** (例如: 发射子弹), 所以用 self.hero 属性保存

后期英雄需要 调用**其他精灵**(例如: 子弹精灵), 所以用 self.heror 属性保存

更新并显示英雄精灵

```
def __update_sprite(self):
    self.hero_group.update()
    self.hero_group.draw(self.screen)
```

移动英雄

英雄精灵主要 通过**x, y 坐标**控制英雄的位置

```
英雄精灵:

def update(self):
        # 移动英雄飞机
        self.rect.x += self.speedx
        self.rect.y += self.speedy
```

主程序 初始化英雄位移 速度

```
主程序:

def __init__(self):
    self.offset = {pygame.K_RIGHT:0, pygame.K_LEFT:0, pygame.K_DOWN:0, pygame.K_UP:0}
```

主程序 需要**获取按键**, 并**判断**是 **水平移动**, 还是**垂直移动**

```
主程序:

事件监听
def __event_monitor(self):
    # 按下键盘
    if event.type == pygame.KEYDOWN:
```

```
        if event.key in self.offset:        # 是否为 方向键
            self.offset[event.key] = 5      # 移动5位

    # 松开键盘
    elif event.type == pygame.KEYUP:

        if event.key in self.offset:        # 是否为 方向键
            self.offset[event.key] = 0      # 移动0位，即不动
```

边框限制

在 **英雄精灵** 限制英雄 的飞行边界, 不能超出屏幕

找到 **移动英雄** 的代码区域

```
英雄精灵:

def update(self):
    # 移动英雄飞机
    self.rect.x += self.speedx
    self.rect.y += self.speedy

    # 边框限制
    if self.rect.x < 0:
        self.rect.x = 0
    elif self.rect.x > SCREEN_RECT.width - self.rect.width:
        self.rect.x = SCREEN_RECT.width - self.rect.width

    if self.rect.y < 0:
        self.rect.y = 0
    elif self.rect.y > SCREEN_RECT.height - self.rect.height:
        self.rect.y = SCREEN_RECT.height - self.rect.height
```

要点分析

x轴

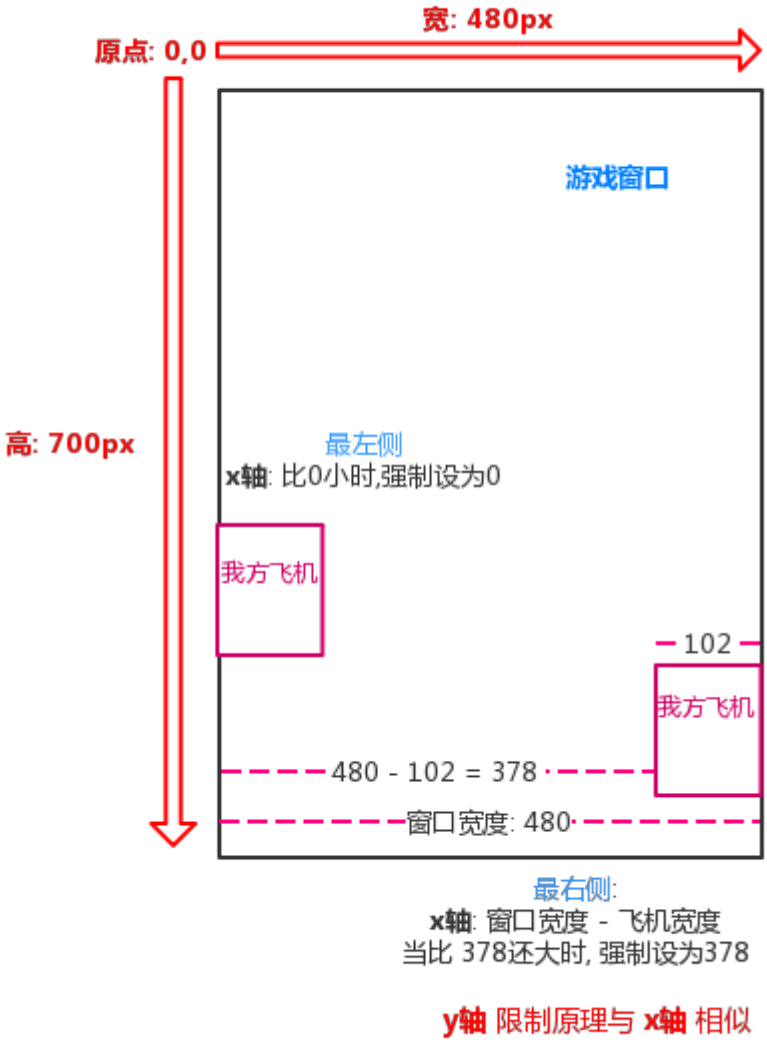
x 为**0** 时, 已经到 最左侧了

x 为 **(屏幕宽-飞机宽)** 时 说明已经到 最右侧了

y轴

y 为**0** 时, 说明已经到 最上方了 (贴着顶部, 不超出屏幕)

y 为 **(屏幕高 - 飞机高)** 时, 说明飞机已经贴到最底部了 (不会超出屏幕)



子弹

- 子弹精灵
- 显示子弹

子弹精灵

```
class Bullet(pygame.sprite.Sprite):
    '''子弹精灵'''
    def __init__(self, image, speed=-5):
        super().__init__()

        # 子弹图像
        self.image = image
```

```
        # 子弹矩形
        self.rect = self.image.get_rect()
        # 子弹速度
        self.speed = speed

    # 移动子弹
    def update(self):
        pass
```

主要

子弹 是**向上**运行的, 所以 speed 默认为 **-5**, 方便子弹在y轴上飞行

显示子弹

- 抠图子弹图片
- 设置发射子弹 定时器
- 创建子弹精灵组
- 定时 发射子弹
- 更新并显示子弹

抠图子弹图片

```
大图
self.plane_img = pygame.image.load('./image/shoot.png')

子弹
self.bullet_rect = pygame.Rect(1004,987,9,21)
self.bullet_img = self.plane_img.subsurface(self.bullet_rect)
```

设置发射子弹 定时器

```
精灵文件

英雄发射子弹事件
HREO_FIRE_EVENT = pygame.USEREVENT + 1
```

注意

pygame.USEREVENT 需要**加1**, 因为前面 敌机事件 已经用过 USEREVENT, 为了避免事件重复, 所有后面所有的自定义事件, 都要加 不同的数字, 来区分不同的事件

```
主程序
def __init__(self):
    # 每 0.2s 创建一个子弹事件
    pygame.time.set_timer(HREO_FIRE_EVENT, 200)
```

创建子弹精灵组

由于 **子弹** 是 **英雄飞机**发射出去的, 所以 **子弹精灵组** 需要再**英雄精灵 初始化时**创建, 方便英雄飞机使用

```
def __init__(self, image):
    self.bullet = pygame.sprite.Group()
```

定时 发射子弹

1. 准备发射子弹的功能
2. 监听 发射子弹事件

1. 准备发射子弹的功能

首先, 给 **英雄精灵** 准备 **发射子弹** 的方法

```
def fire(self, bullet_img):
    # 创建子弹精灵
    bullet1 = Bullet(bullet_img)

    # 设置 子弹初始位置
    bullet1.rect.centerx = self.rect.centerx
    bullet1.rect.y = self.rect.y

    # 将 子弹添加进精灵组
    self.bullet.add(bullet1)
```

注意

- 这里参数 多个一个 bullet_img
    - **原因: 子弹图像** 是在主程序中的, **无法直接**在精灵类中**获取**, 所以在**发射子弹前**, 需要先将 **子弹图像 传过来**
- 每创建一个子弹精灵, 就**添加进 精灵组**

2. 监听 发射子弹 事件

```
def __event_monitor(self):
    if event.type == HREO_FIRE_EVENT:
        self.hero.fire(self.bullet_img)
```

注意

根据当初 设定的 **发射子弹事件** 来监听, 一旦发生, 则调用 **英雄飞机** 的 **发射子弹**功能

别忘记了, 这里的 **发射子弹** 功能需要 **子弹图像**

更新并显示子弹

```
def __update_sprite(self):
        self.hero.bullet.update()
        self.hero.bullet.draw(self.screen)
```

注意

因为 **子弹精灵组** 在 **英雄精灵** 中, 所以需要通过 **英雄精灵** 来更新和绘制

碰撞

pygame提供 **碰撞检测**的方法

- pygame.sprite.groupcollide()
- pygame.sprite.spritecollide()

pygame.sprite.groupcollide(group1, group2, kill1, kill2, collided)

- 功能
  - 检测 **两个精灵组** 中 **精灵** 是否碰撞
- 参数
  - group1 精灵组1
  - group2 精灵组2
  - kill1 值: bool True: 删除精灵组1
  - kill2 值: bool True: 删除精灵组2
  - collided 值: none 计算碰撞的回调函数
    - 若没有指定, 那么每个精灵必须要有 rect 属性
- 返回值
  - Sprite_list 精灵列表

pygame.sprite.spritecollide(sprite group, kill, collided)

- 功能
  - 检测 **精灵** 和 **精灵组** 是否碰撞
- 参数
  - sprite 精灵
  - group 精灵组

    - kill 值: bool True: 删除精灵组
    - collided 值: none 计算碰撞的回调函数
        - 若没有指定, 那么每个精灵必须要有 rect 属性
- 返回值
    - Sprite_list 精灵列表

子弹碰撞敌机

```
def __collision(self):
        pygame.sprite.groupcollide(self.hero.bullet, self.enemy_group, True, True)
```

注意

当 **英雄飞机的子弹** 和 **敌机** 碰撞时, 两个都应该销毁

所以, kill 都设置为true

敌机碰撞英雄

```
def __collision(self):
    enemy_list = pygame.sprite.spritecollide(self.hero, self.enemy_group, True)
        # 如果列表有值，说明英雄和敌机已经碰撞
        if len(enemy_list) > 0:
            # 杀死英雄
            self.hero.kill()
            # 结束游戏
            self.__game_over()
```

注意

当 **英雄飞机** 和 **敌机精灵组** 碰撞时, 英雄应该被**摧毁**

但是, 方法第一个只能写 **精灵**, 第二个参数只能写 **精灵组**, 只能导致碰撞后, 只有**精灵组(敌机)被摧毁**, 而**英雄没有被摧毁**

此时, 该方法返回的是 精灵列表, 每一个被摧毁的敌机, 都会送进 **sprite_list** , 并返回.

那么, 此时**检测** sprite_list **是否有值**, 若有, 则**证明有敌机被摧毁**了, 也**证明 英雄被敌机碰撞**了, 那么此时可以**移除 英雄精灵**. 从而达到摧毁英雄的效果.

英雄一旦被摧毁, 也可以**结束游戏**