[笔记][黑马 Python 之 Python 飞机大战 - 2]

Python

---

027. 背景图像-01-交替滚动实现思路分析

目标

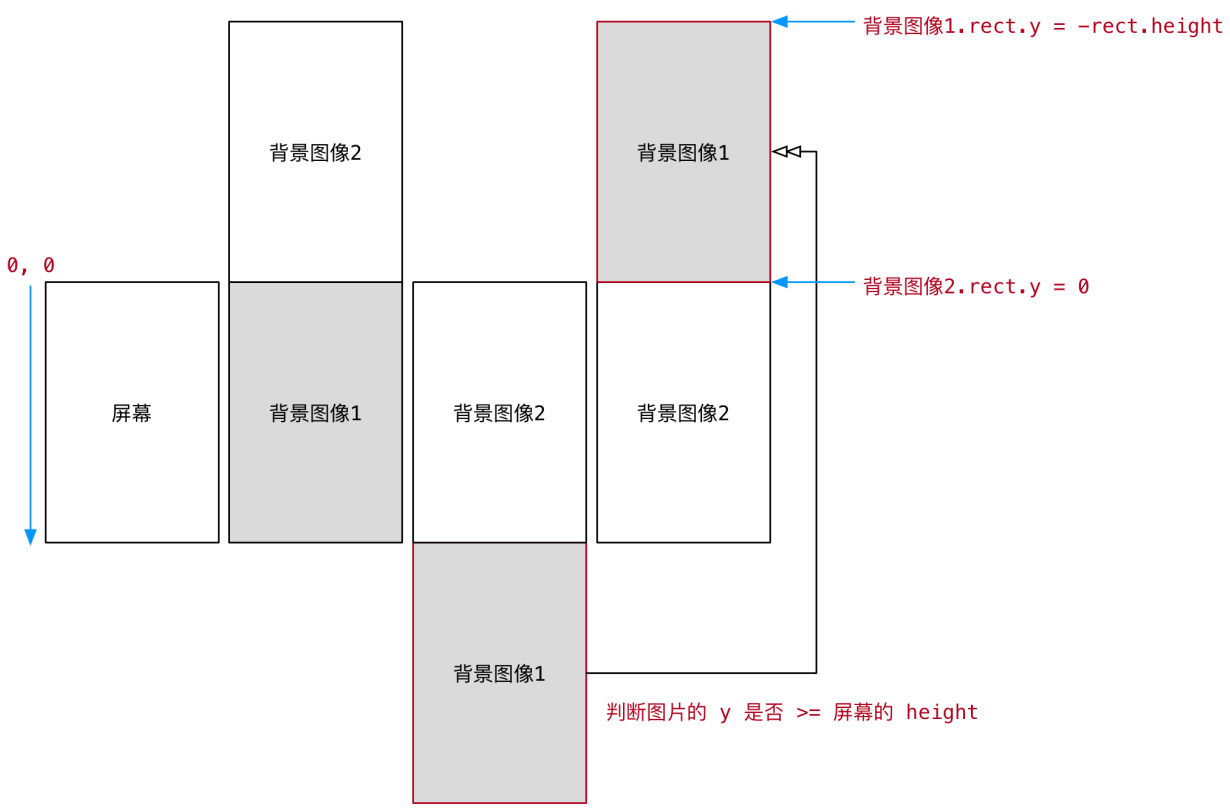
- 背景交替滚动的思路确定
- 显示游戏背景

背景交替滚动的思路确定

运行 **备课代码**，**观察** 背景图像的显示效果：

- 游戏启动后，**背景图像** 会 **连续不断地 向下方** 移动
- 在 **视觉上** 产生英雄的飞机不断向上方飞行的 **错觉** —— 在很多跑酷类游戏中常用的套路
    - **游戏的背景** 不断变化
    - **游戏的主角** 位置保持不变

实现思路分析



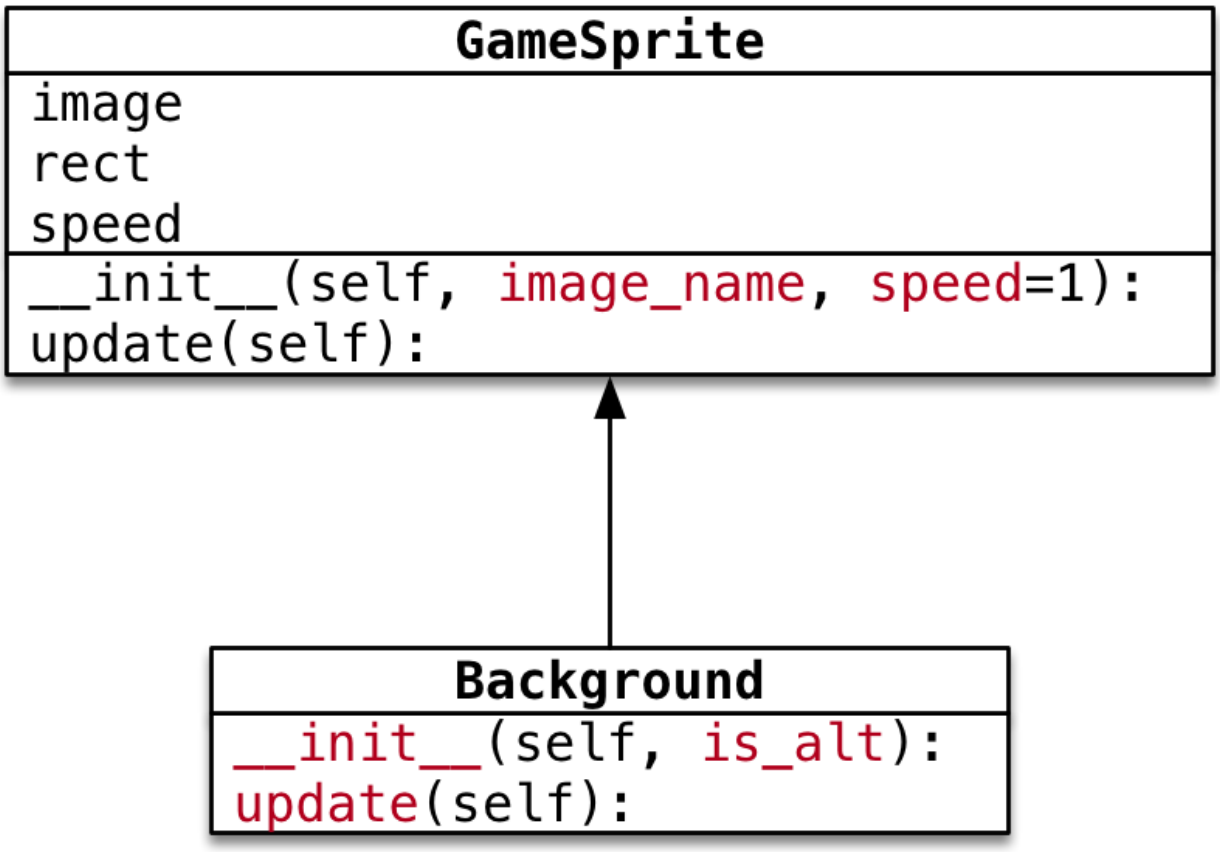
1. 创建两张背景图像精灵
    - 第 `1` 张 **完全和屏幕重合**
    - 第 `2` 张在 **屏幕的正上方**

2. 两张图像 **一起向下方运动**
    - `self.rect.y += self.speed`

3. 当 **任意背景精灵** 的 `rect.y >= 屏幕的高度` 说明已经 **移动到屏幕下方**
4. 将 **移动到屏幕下方的这张图像** 设置到 **屏幕的正上方**
    - `rect.y = -rect.height`

---

028. 背景图像-02-背景类的设计与基本实现



- **初始化方法**
  - 直接指定 **背景图片**
  - `is_alt` 判断是否是另一张图像
  - `False` 表示 **第一张图像**，需要与屏幕重合
  - `True` 表示 **另一张图像**，在屏幕的正上方
- **update()** 方法
  - 判断 **是否移动出屏幕**，如果是，将图像设置到 **屏幕的正上方**，从而实现 **交替滚动**

> **继承** 如果父类提供的方法，不能满足子类的需求：
>
> - 派生一个子类
> - 在子类中针对特有的需求，重写父类方法，并且进行扩展

```python
import pygame

# 屏幕大小的常量
SCREEN_RECT = pygame.Rect(0, 0, 480, 700)
# 刷新的帧率
FRAME_PER_SEC = 60


class GameSprite(pygame.sprite.Sprite):
    """飞机大战游戏精灵"""

    def __init__(self, image_name, speed=1):
        # 调用父类的初始化方法
```

```
        # super(GameSprite, self).__init__()
        super().__init__()

        # 定义对象的属性
        self.image = pygame.image.load(image_name)
        self.rect = self.image.get_rect()
        self.speed = speed

    def update(self):
        # 在屏幕的垂直方向上移动
        self.rect.y += self.speed


class Background(GameSprite):
    """游戏背景精灵"""
    def update(self):
        # 1. 调用父类的方法实现
        super().update()
        # 2. 判断是否移出屏幕
        # 如果移出屏幕，将图像设置到屏幕上方
        if self.rect.y >= SCREEN_RECT.height:
            self.rect.y = -self.rect.height
```

029. 背景图像-03-背景图像的交替滚动实现

在 `plane_main.py` 中显示背景精灵

1. 在 `__create_sprites` 方法中创建 **精灵** 和 **精灵组**
2. 在 `__update_sprites` 方法中，让 **精灵组** 调用 `update()` 和 `draw()` 方法

> `__create_sprites` 方法

```
def __create_sprites(self):

    # 创建背景精灵和精灵组
    bg1 = Background("./images/background.png")
    bg2 = Background("./images/background.png")
    bg2.rect.y = -bg2.rect.height

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

> `__update_sprites` 方法

```
def __update_sprites(self):

    self.back_group.update()
    self.back_group.draw(self.screen)
```

---

030. 背景图像-04-利用初始化方法简化背景精灵的创建

利用初始化方法，简化背景精灵创建

> 思考 —— 上一小结完成的代码存在什么样的问题？能否简化？

- 在主程序中，创建的**两个背景精灵**，**传入了相同的图像文件路径**
- 创建 **第二个 背景精灵** 时，**在主程序中**，设置背景精灵的图像位置

> 思考 —— 精灵 **初始位置** 的设置，应该 **由主程序负责**？还是 **由精灵自己负责**？

答案 —— 由精灵自己负责

- 根据面向对象设计原则，应该将对象的职责，封装到类的代码内部
- 尽量简化程序调用一方的代码调用

初始化方法

- 直接指定 **背景图片**
- `is_alt` 判断是否是另一张图像
  - `False` 表示 **第一张图像**，需要与屏幕重合
  - `True` 表示 **另一张图像**，在屏幕的正上方

在 `plane_sprites.py` 中实现 `Background` 的 **初始化方法**

```
def __init__(self, is_alt=False):

    image_name = "./images/background.png"
    super().__init__(image_name)

    # 判断是否交替图片，如果是，将图片设置到屏幕顶部
    if is_alt:
```

```
            self.rect.y = -self.rect.height
```

- 修改 `plane_main` 的 `__create_sprites` 方法

```
# 创建背景精灵和精灵组
bg1 = Background()
bg2 = Background(True)

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

---

031. 敌机-01-定时器语法介绍

运行 **备课代码**，**观察** 敌机的 **出现规律**：

1. 游戏启动后，**每隔 1 秒** 会 **出现一架敌机**
2. 每架敌机 **向屏幕下方飞行**，飞行 **速度各不相同**
3. 每架敌机出现的 **水平位置** 也不尽相同
4. 当敌机 **从屏幕下方飞出**，不会再飞回到屏幕中

定时器

- 在 `pygame` 中可以使用 `pygame.time.set_timer()` 来添加 **定时器**
- 所谓 **定时器**，就是 **每隔一段时间**，去 **执行一些动作**

```
set_timer(eventid, milliseconds) -> None
```

- `set_timer` 可以创建一个 **事件**
- 可以在 **游戏循环** 的 **事件监听** 方法中捕获到该事件
- 第 1 个参数 **事件代号** 需要基于常量 `pygame.USEREVENT` 来指定
    - `USEREVENT` 是一个整数，再增加的事件可以使用 `USEREVENT + 1` 指定，依次类推...
- 第 2 个参数是 **事件触发** 间隔的 **毫秒值**

定时器事件的监听

- 通过 `pygame.event.get()` 可以获取当前时刻所有的 **事件列表**
- **遍历列表** 并且判断 `event.type` 是否等于 `eventid`，如果相等，表示 **定时器事件** 发生

---

032. 敌机-02-定义并且监听创建敌机的定时器事件

`pygame` 的 **定时器** 使用套路非常固定：

1. 定义 **定时器常量** —— `eventid`
2. 在 **初始化方法** 中，调用 `set_timer` 方法 **设置定时器事件**
3. 在 **游戏循环** 中，**监听定时器事件**

---

1) 定义事件

- 在 `plane_sprites.py` 的顶部定义 **事件常量**

```
# 敌机的定时器事件常量
CREATE_ENEMY_EVENT = pygame.USEREVENT
```

- 在 `PlaneGame` 的 **初始化方法** 中 **创建用户事件**

```
# 4. 设置定时器事件 - 每秒创建一架敌机
pygame.time.set_timer(CREATE_ENEMY_EVENT, 1000)
```

2) 监听定时器事件

- 在 `__event_handler` 方法中增加以下代码：

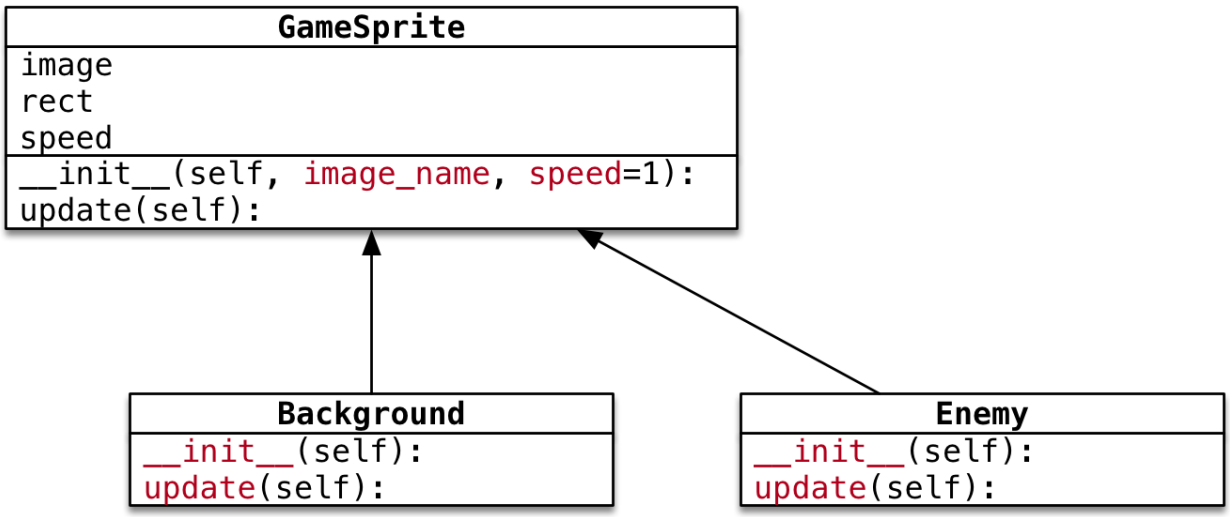
```
def __event_handler(self):

    for event in pygame.event.get():

        # 判断是否退出游戏
        if event.type == pygame.QUIT:
            PlaneGame.__game_over()
        elif event.type == CREATE_ENEMY_EVENT:
            print("敌机出场...")
```

---

033. 敌机-03-设计并准备敌机类

1. 游戏启动后，**每隔 1 秒** 会 **出现一架敌机**
2. 每架敌机 **向屏幕下方飞行**，飞行 **速度各不相同**
3. 每架敌机出现的 **水平位置** 也不尽相同

4. 当敌机 **从屏幕下方飞出**，不会再飞回到屏幕中



- **初始化方法**
  - 指定 **敌机图片**
  - **随机** 敌机的 **初始位置** 和 **初始速度**
- 重写 **update()** 方法
  - 判断 **是否飞出屏幕**，如果是，从 **精灵组** 删除

敌机类的准备

- 在 `plane_sprites` 新建 `Enemy` 继承自 `GameSprite`
- 重写 **初始化方法**，直接指定 **图片名称**
- 暂时 **不实现 随机速度** 和 **随机位置** 的指定
- 重写 `update` 方法，判断是否飞出屏幕

```
class Enemy(GameSprite):
    """敌机精灵"""

    def __init__(self):

        # 1. 调用父类方法，创建敌机精灵，并且指定敌机的图像
        super().__init__("./images/enemy1.png")

        # 2. 设置敌机的随机初始速度

        # 3. 设置敌机的随机初始位置

    def update(self):

        # 1. 调用父类方法，让敌机在垂直方向运动
        super().update()

        # 2. 判断是否飞出屏幕，如果是，需要将敌机从精灵组删除
        if self.rect.y >= SCREEN_RECT.height:
```

```
            print("敌机飞出屏幕...")
```

034. 敌机-04-定时创建并显示敌机精灵

演练步骤

1. 在 `__create_sprites`，添加 **敌机精灵组**
    - 敌机是 **定时被创建的**，因此在初始化方法中，不需要创建敌机
2. 在 `__event_handler`，创建敌机，并且 **添加到精灵组**
    - 调用 **精灵组** 的 `add` 方法可以 **向精灵组添加精灵**
3. 在 `__update_sprites`，让 **敌机精灵组** 调用 `update` 和 `draw` 方法

精灵（需要派生子类）
`image` 记录图像数据
`rect` 记录在屏幕上的位置
`update(*args)`：更新精灵位置
`kill()`：从所有组中删除

精灵组
`__init__(self, *精灵)`：
`add(*sprites)`：向组中增加精灵
`sprites()`：返回所有精灵列表
`update(*args)`：让组中所有精灵调用 `update` 方法
`draw(Surface)`：将组中所有精灵的 `image`，绘制到 `Surface` 的 `rect` 位置

演练代码

- 修改 `plane_main` 的 `__create_sprites` 方法

```
# 敌机组
self.enemy_group = pygame.sprite.Group()
```

- 修改 `plane_main` 的 `__update_sprites` 方法

```
self.enemy_group.update()
self.enemy_group.draw(self.screen)
```

- 定时出现敌机

```
elif event.type == CREATE_ENEMY_EVENT:
    self.enemy_group.add(Enemy())
```

035. 敌机-05-随机位置以及随机速度

导入模块

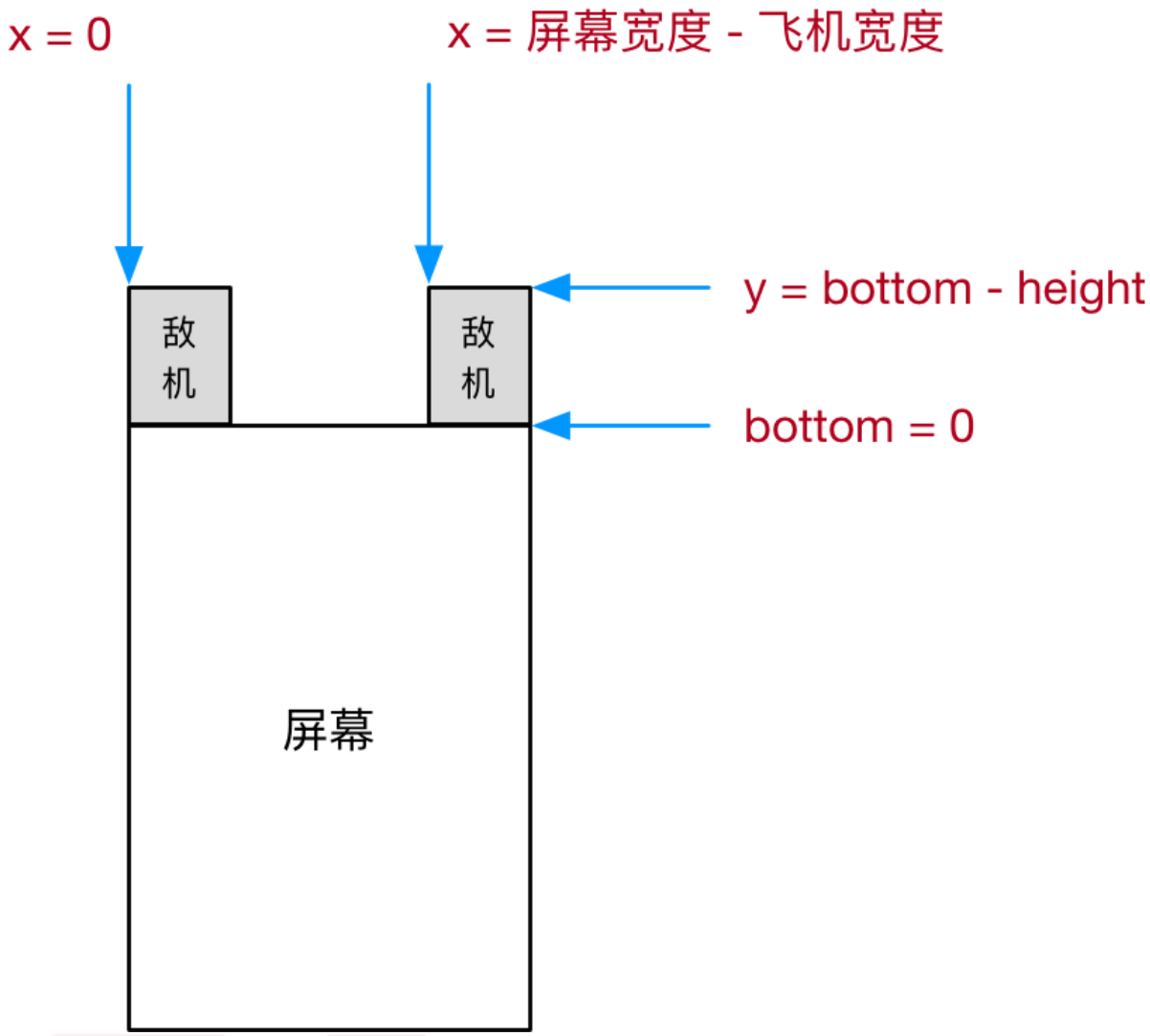
- 在导入模块时，**建议** 按照以下顺序导入

```
1. 官方标准模块导入
2. 第三方模块导入
3. 应用程序模块导入
```

- 修改 `plane_sprites.py` 增加 `random` 的导入

```
import random
```

随机位置



使用 `pygame.Rect` 提供的 `bottom` 属性，在指定敌机初始位置时，会比较方便

- `bottom = y + height`
- `y = bottom - height`

代码实现

- 修改 **初始化方法**，随机敌机出现 **速度** 和 **位置**

```
def __init__(self):

    # 1. 调用父类方法，创建敌机精灵，并且指定敌机的图像
    super().__init__("./images/enemy1.png")

    # 2. 设置敌机的随机初始速度 1 ~ 3
    self.speed = random.randint(1, 3)

    # 3. 设置敌机的随机初始位置
    self.rect.bottom = 0

    max_x = SCREEN_RECT.width - self.rect.width
    self.rect.x = random.randint(0, max_x)
```

---

036. 敌机-06-销毁飞出屏幕的敌机

- 敌机移出屏幕之后，如果 **没有撞到英雄**，敌机的历史使命已经终结
- 需要从 **敌机组** 删除，否则会造成 **内存浪费**

检测敌机被销毁

- `__del__` 内置方法会在对象被销毁前调用，在开发中，可以用于 **判断对象是否被销毁**

```
def __del__(self):
    print("敌机挂了 %s" % self.rect)
```

代码实现

- 判断敌机是否飞出屏幕，如果是，调用 `kill()` 方法从所有组中删除

```
def update(self):
    super().update()

    # 判断敌机是否移出屏幕
    if self.rect.y >= SCREEN_RECT.height:
        # 将精灵从所有组中删除
        self.kill()
```

---

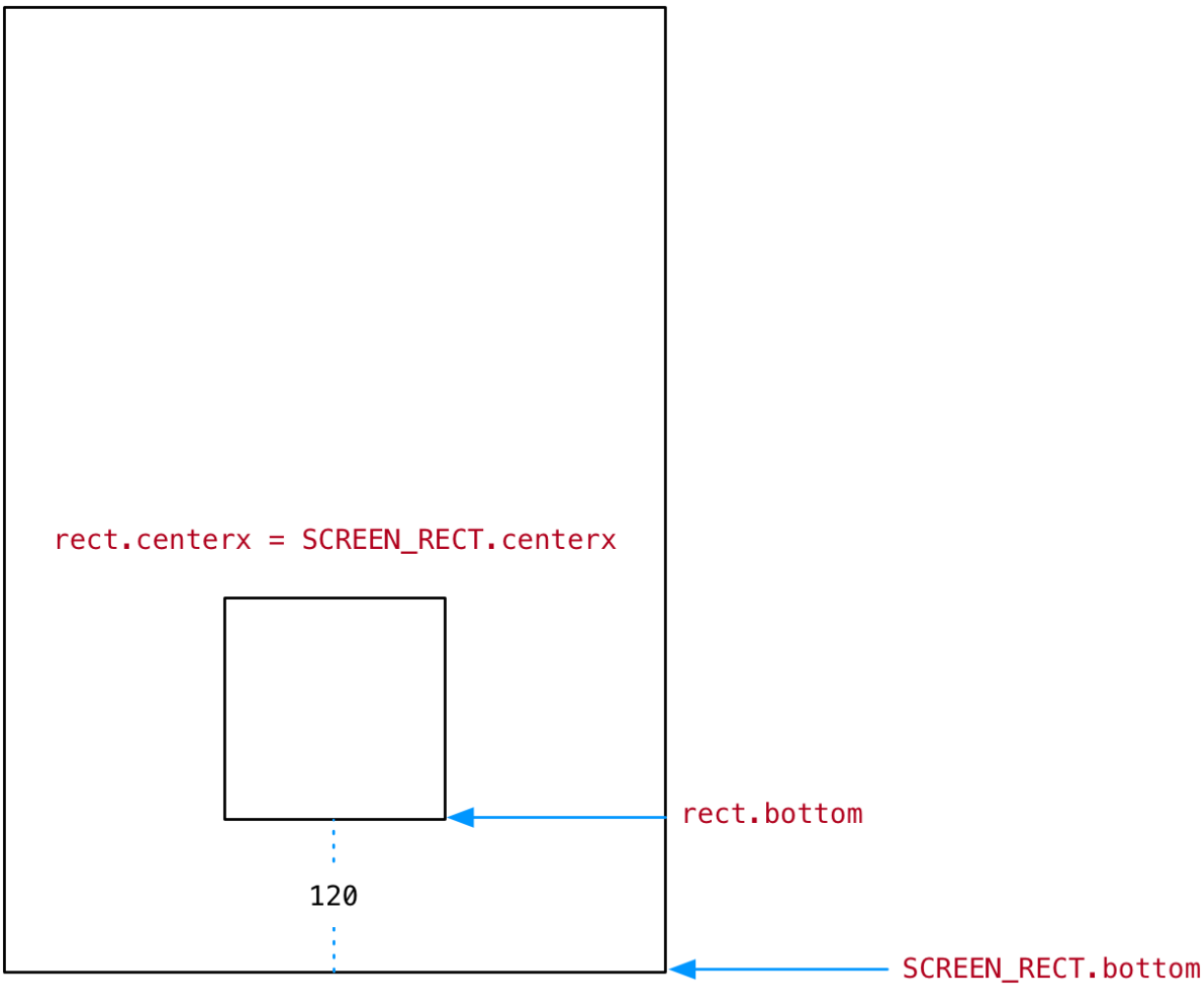
037. 英雄-01-需求分析和类设计

目标

- 设计 **英雄** 和 **子弹** 类
- 使用 `pygame.key.get_pressed()` 移动英雄
- 发射子弹

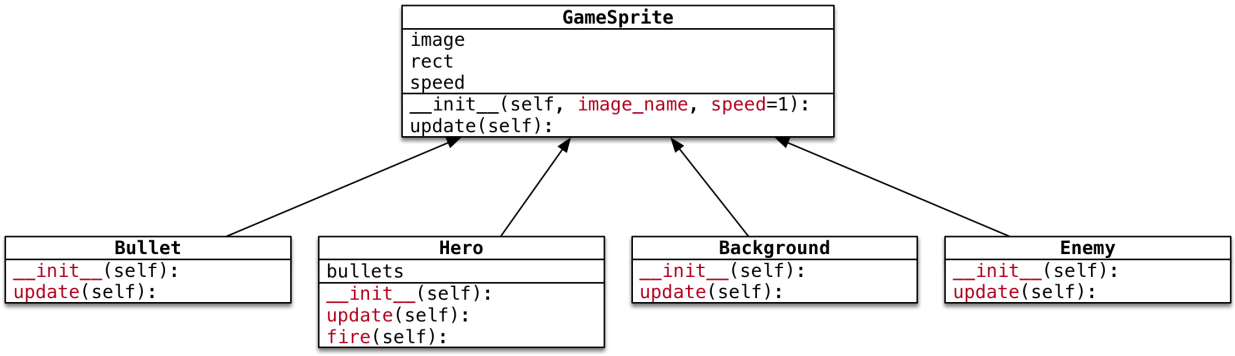
英雄需求

1. 游戏启动后，**英雄** 出现在屏幕的 **水平中间** 位置，距离 **屏幕底部** `120` **像素**
2. **英雄** 每隔 `0.5` 秒发射一次子弹，每次 **连发三枚子弹**
3. **英雄** 默认不会移动，需要通过 **左/右** 方向键，控制 **英雄** 在水平方向移动



子弹需求

1. **子弹** 从 **英雄** 的正上方发射 **沿直线** 向 **上方** 飞行
2. **飞出屏幕后**，需要从 **精灵组** 中删除



Hero —— 英雄

- **初始化方法**
  - 指定 **英雄图片**
  - **初始速度 = 0** —— 英雄默认静止不动
  - 定义 `bullets` **子弹精灵组** 保存子弹精灵
- 重写 **update()** 方法
  - 英雄需要 **水平移动**
  - 并且需要保证不能 **移出屏幕**
- 增加 `bullets` 属性，记录所有 **子弹精灵**
- 增加 `fire` 方法，用于发射子弹

Bullet —— 子弹

- **初始化方法**
  - 指定 **子弹图片**
  - **初始速度 = -2** —— 子弹需要向上方飞行
- 重写 **update()** 方法
  - 判断 **是否飞出屏幕**，如果是，从 **精灵组** 删除

---

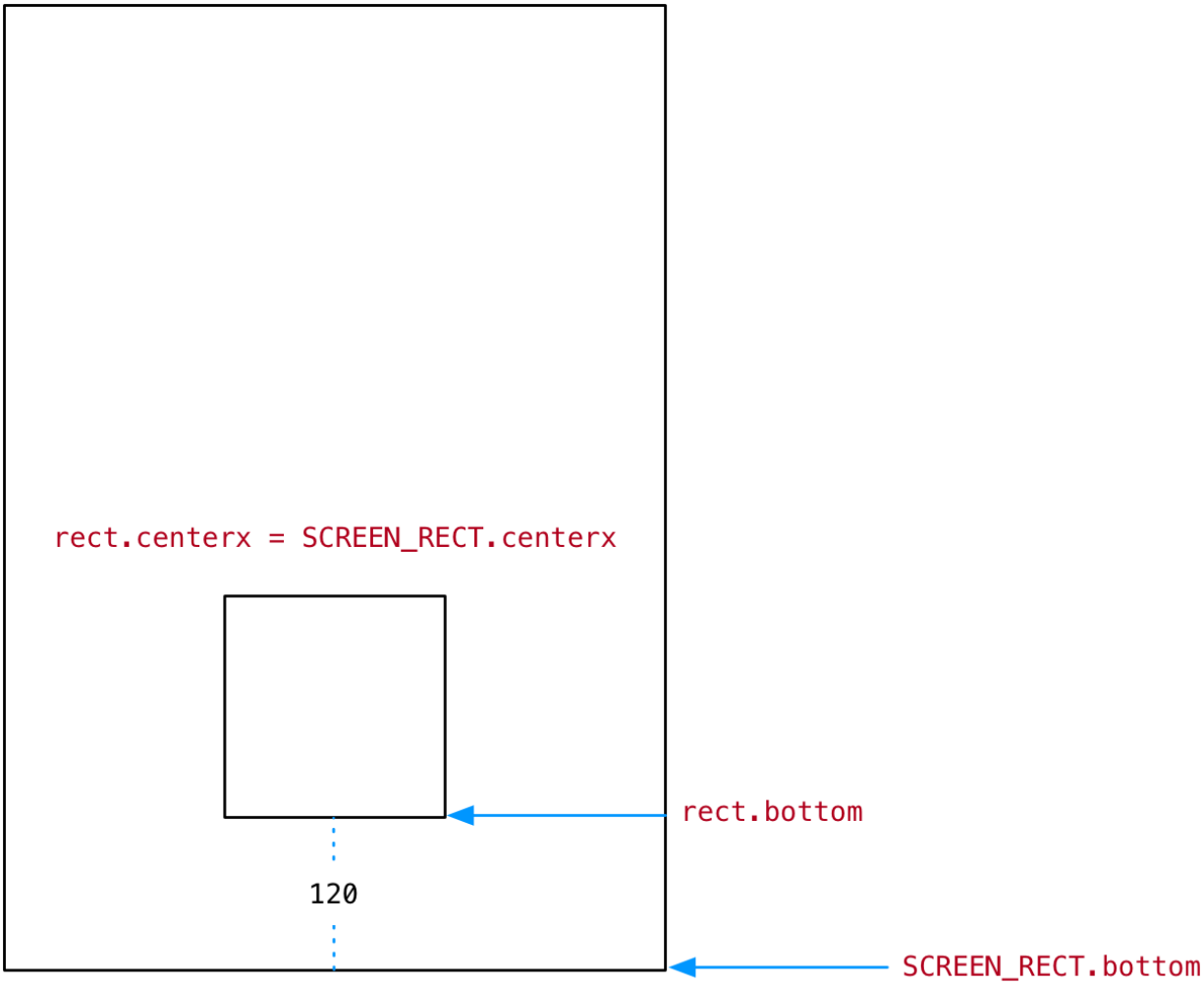
038. 英雄-02-准备英雄类

准备英雄类

- 在 `plane_sprites` 新建 `Hero` 类
- 重写 **初始化方法**，直接指定 **图片名称**，并且将初始速度设置为 `0`
- 设置 **英雄的初始位置**

pygame.Rect
x, y, left, top, bottom, right, center, centerx, centery, size, width, height

- `centerx = x + 0.5 * width`
- `centery = y + 0.5 * height`
- `bottom = y + height`




```python
class Hero(GameSprite):
    """英雄精灵"""

    def __init__(self):

        super().__init__("./images/me1.png", 0)

        # 设置初始位置
        self.rect.centerx = SCREEN_RECT.centerx
        self.rect.bottom = SCREEN_RECT.bottom - 120
```

039. 英雄-03-绘制英雄

绘制英雄

1. 在 `__create_sprites` ，添加 **英雄精灵** 和 **英雄精灵组**
    - 后续要针对 **英雄** 做 **碰撞检测** 以及 **发射子弹**
    - 所以 **英雄** 需要 **单独定义成属性**
2. 在 `__update_sprites` ，让 **英雄精灵组** 调用 `update` 和 `draw` 方法

代码实现

- 修改 `__create_sprites` 方法如下：

```
# 英雄组
self.hero = Hero()
self.hero_group = pygame.sprite.Group(self.hero)
```

- 修改 `__update_sprites` 方法如下：

```
self.hero_group.update()
self.hero_group.draw(self.screen)
```

---

040. 英雄-04-两种按键方式对比

> 在 `pygame` 中针对 **键盘按键的捕获**，有 **两种** 方式

- **第一种方式** 判断 `event.type == pygame.KEYDOWN`
- **第二种方式**
    1. 首先使用 `pygame.key.get_pressed()` 返回 **所有按键元组**
    2. 通过 **键盘常量**，判断元组中 **某一个键是否被按下** —— 如果被按下，对应数值为 `1`

提问 这两种方式之间有什么区别呢？

- 第一种方式

```
elif event.type == pygame.KEYDOWN and event.key == pygame.K_RIGHT:
    print("向右移动...")
```

- 第二种方式

```
# 返回所有按键的元组，如果某个键被按下，对应的值会是1
keys_pressed = pygame.key.get_pressed()
# 判断是否按下了方向键
if keys_pressed[pygame.K_RIGHT]:
    print("向右移动...")
```

结论

- **第一种方式** `event.type` 用户 **必须要抬起按键** 才算一次 **按键事件**，**操作灵活性会大打折扣**
- **第二种方式** 用户可以按住方向键不放，就能够实现持续向某一个方向移动了，**操作灵活性更好**

---

041. 英雄-05-控制英雄左右移动

移动英雄位置

演练步骤

1. 在 `Hero` 类中重写 `update` 方法
    - 用 **速度** `speed` 和 **英雄** `rect.x` 进行叠加
    - **不需要调用父类方法** —— 父类方法只是实现了单纯的垂直运动
2. 在 `__event_handler` 方法中根据 **左右方向键** 设置英雄的 **速度**
    - **向右** => `speed = 2`
    - **向左** => `speed = -2`
    - **其他** => `speed = 0`

代码演练

- 在 `Hero` 类，重写 `update()` 方法，**根据速度水平移动** 英雄的飞机

```
def update(self):

    # 飞机水平移动
    self.rect.x += self.speed
```

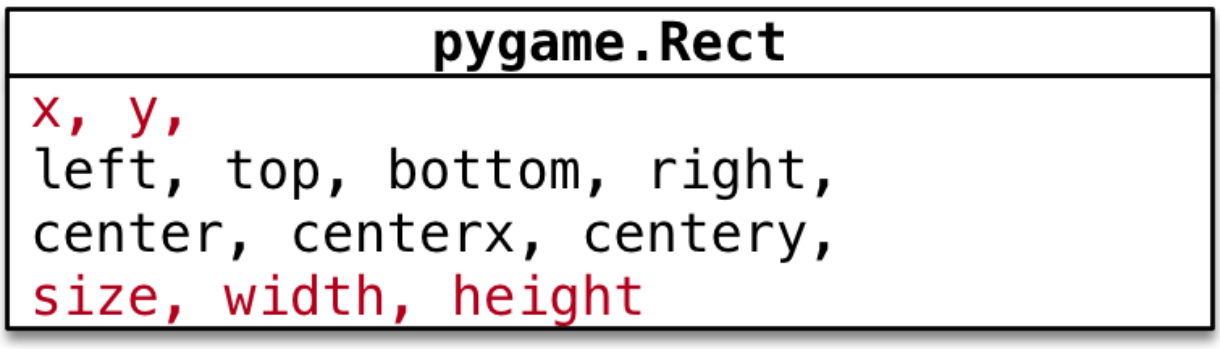
- 调整键盘按键代码

```
# 获取用户按键
keys_pressed = pygame.key.get_pressed()

if keys_pressed[pygame.K_RIGHT]:
    self.hero.speed = 2
```

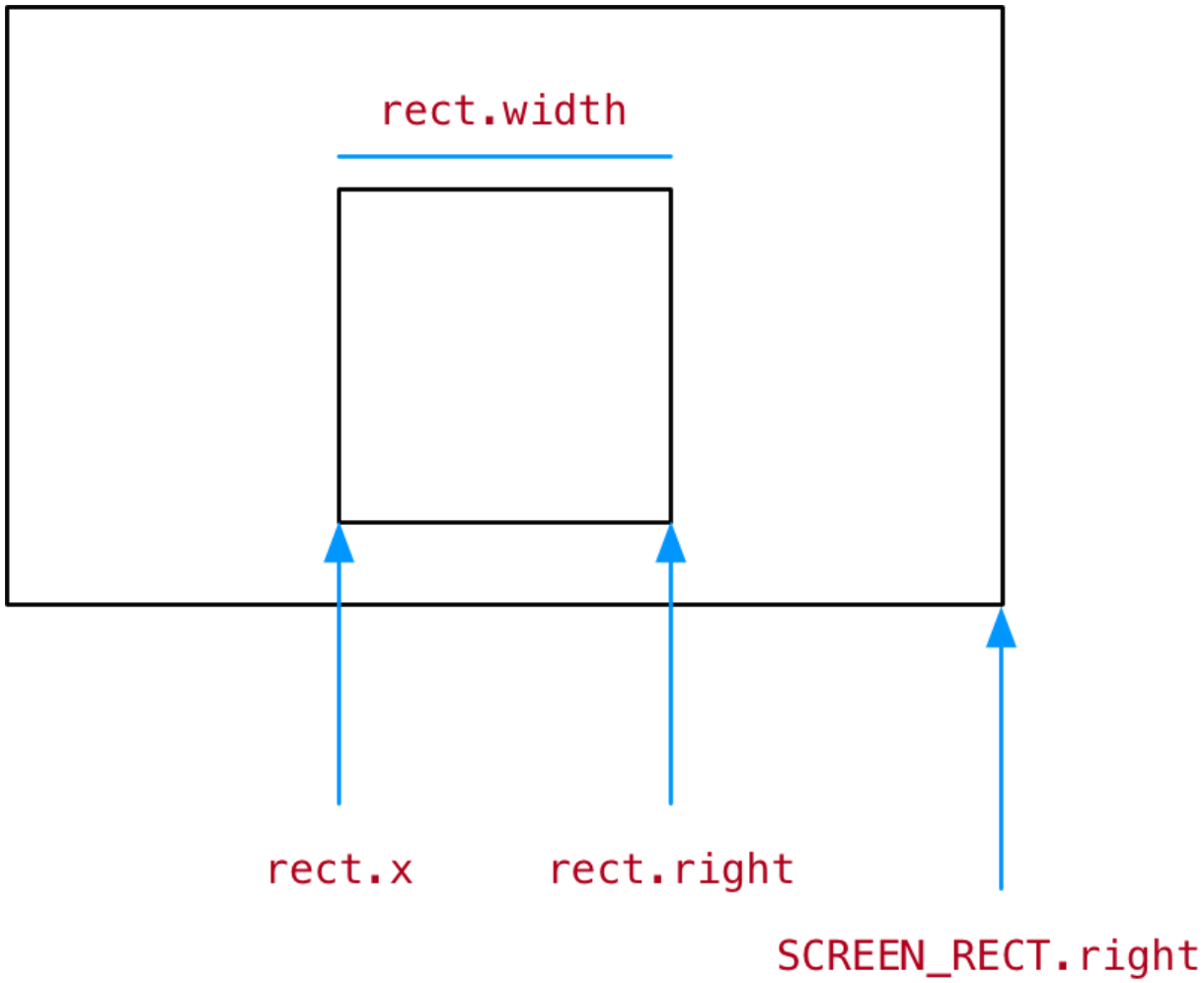
```
    elif keys_pressed[pygame.K_LEFT]:
        self.hero.speed = -2
    else:
        self.hero.speed = 0
```

042. 英雄-06-英雄边界控制

- 在 `Hero` 类的 `update()` 方法判断 **英雄** 是否超出 **屏幕边界**



- `right = x + width` 利用 `right` 属性可以非常容易的针对右侧设置精灵位置



```
def update(self):

    # 飞机水平移动
    self.rect.x += self.speed

    # 判断屏幕边界
    if self.rect.left < 0:
        self.rect.left = 0
    if self.rect.right > SCREEN_RECT.right:
        self.rect.right = SCREEN_RECT.right
```

---

043. 发射子弹-01-添加并监听英雄发射子弹事件

需求回顾 —— 英雄需求

1. 游戏启动后，**英雄** 出现在屏幕的 **水平中间** 位置，距离 **屏幕底部** `120` **像素**
2. **英雄** 每隔 `0.5` 秒发射一次子弹，每次 **连发三枚子弹**
3. **英雄** 默认不会移动，需要通过 **左/右** 方向键，控制 **英雄** 在水平方向移动

添加发射子弹事件

`pygame` 的 **定时器** 使用套路非常固定：

1. 定义 **定时器常量** —— `eventid`
2. 在 **初始化方法** 中，调用 `set_timer` 方法 **设置定时器事件**
3. 在 **游戏循环** 中，**监听定时器事件**

代码实现

- 在 `Hero` 中定义 `fire` 方法

```
def fire(self):
    print("发射子弹...")
```

- 在 `plane_main.py` 的顶部定义 **发射子弹** 事件常量

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

- 在 `__init__` 方法末尾中添加 **发射子弹** 事件

```
# 每隔 0.5 秒发射一次子弹
pygame.time.set_timer(HERO_FIRE_EVENT, 500)
```

- 在 `__event_handler` 方法中让英雄发射子弹

```
elif event.type == HERO_FIRE_EVENT:
    self.hero.fire()
```

---

044. 发射子弹-02-定义子弹类

需求回顾 —— 子弹需求

1. **子弹** 从 **英雄** 的正上方发射 **沿直线** 向 **上方** 飞行
2. **飞出屏幕后**，需要从 **精灵组** 中删除

Bullet —— 子弹

- **初始化方法**
  - 指定 **子弹图片**
  - **初始速度 = -2** —— 子弹需要向上方飞行
- 重写 **update()** 方法
  - 判断 **是否飞出屏幕**，如果是，从 **精灵组** 删除

定义子弹类

- 在 `plane_sprites` 新建 `Bullet` 继承自 `GameSprite`
- 重写 **初始化方法**，直接指定 **图片名称**，并且设置 **初始速度**
- 重写 `update()` 方法，判断子弹 **飞出屏幕从精灵组删除**

```
class Bullet(GameSprite):
    """子弹精灵"""

    def __init__(self):

        super().__init__("./images/bullet1.png", -2)

    def update(self):
```

```
        super().update()

        # 判断是否超出屏幕，如果是，从精灵组删除
        if self.rect.bottom < 0:
            self.kill()
```

---

045. 发射子弹-03-发射子弹

演练步骤

1. 在 `Hero` 的 **初始化方法** 中创建 **子弹精灵组** 属性
2. 修改 `plane_main.py` 的 `__update_sprites` 方法，让 **子弹精灵组** 调用 `update` 和 `draw` 方法
3. 实现 `fire()` 方法
    - 创建子弹精灵
    - 设置初始位置 —— 在 **英雄的正上方**
    - 将 **子弹** 添加到精灵组

代码实现

- 初始化方法

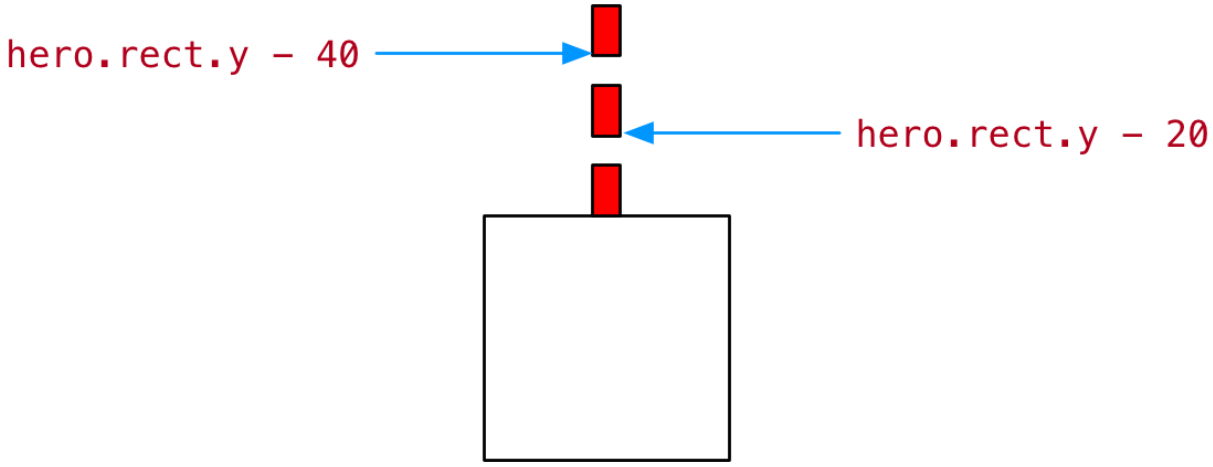
```
# 创建子弹的精灵组
self.bullets = pygame.sprite.Group()
```

- 修改 `fire()` 方法

```
def fire(self):

    # 1. 创建子弹精灵
    bullet = Bullet()

    # 2. 设置精灵的位置
    bullet.rect.bottom = self.rect.y - 20
    bullet.rect.centerx = self.rect.centerx

    # 3. 将精灵添加到精灵组
    self.bullets.add(bullet)
```

---

046. 发射子弹-04-一次发射三枚子弹



- 修改 `fire()` 方法，一次发射三枚子弹

```
def fire(self):

    for i in (1, 2, 3):
        # 1. 创建子弹精灵
        bullet = Bullet()

        # 2. 设置精灵的位置
        bullet.rect.bottom = self.rect.y - i * 20
        bullet.rect.centerx = self.rect.centerx

        # 3. 将精灵添加到精灵组
        self.bullets.add(bullet)
```

> 【!】注意可以用 `for i in range(3):`

---

047. 碰撞检测-01-子弹摧毁敌机

- `pygame` 提供了 **两个非常方便** 的方法可以实现碰撞检测：

`pygame.sprite.groupcollide()`

- **两个精灵组** 中 **所有的精灵** 的碰撞检测

```
groupcollide(group1, group2, dokill1, dokill2, collided = None) -> Sprite
_dict
```

- 如果将 `dokill` 设置为 `True`，则 **发生碰撞的精灵将被自动移除**
- `collided` 参数是用于 **计算碰撞的回调函数**

- 如果没有指定，则每个精灵必须有一个 `rect` 属性

---

048. 碰撞检测-02-敌机摧毁英雄

`pygame.sprite.spritecollide()`

- 判断 **某个精灵** 和 **指定精灵组** 中的精灵的碰撞

```
spritecollide(sprite, group, dokill, collided = None) -> Sprite_list
```

- 如果将 `dokill` 设置为 `True` ，则 **指定精灵组** 中 **发生碰撞的精灵将被自动移除**
- `collided` 参数是用于 **计算碰撞的回调函数**
  - 如果没有指定，则每个精灵必须有一个 `rect` 属性
- 返回 **精灵组** 中跟 **精灵** 发生碰撞的 **精灵列表**

碰撞实现

```
def __check_collide(self):

    # 1. 子弹摧毁敌机
    pygame.sprite.groupcollide(self.hero.bullets, self.enemy_group, True,
 True)

    # 2. 敌机撞毁英雄
    enemies = pygame.sprite.spritecollide(self.hero, self.enemy_group, Tr
ue)

    # 判断列表时候有内容
    if len(enemies) > 0:

        # 让英雄牺牲
        self.hero.kill()

        # 结束游戏
        PlaneGame.__game_over()
```

完成于 `201810102045`