

1 项目实战 —— 飞机大战

目标

- 强化 **面向对象** 程序设计
- 体验使用 **pygame** 模块进行 **游戏开发**

实战步骤

1. **pygame** 快速体验
2. **飞机大战** 实战

确认模块 —— **pygame**

- **pygame** 就是一个 Python 模块，专为电子游戏设计
- 官方网站: <https://www.pygame.org/>
 - **提示**: 要学习第三方模块，通常最好的参考资料就在官方网站

网站栏目	内容
GettingStarted	在各平台安装模块的说明
Docs	pygame 模块所有 类 和 子类 的参考手册

安装 **pygame**

```
$ sudo pip3 install pygame
```

Windows 安装

```
pip install pygame
```

验证安装

```
$ python -m pygame.examples.aliens
```

2 pygame 快速入门

目标

- 项目准备
- 使用 `pygame` 创建图形窗口
- 理解 **图像** 并实现图像绘制
- 理解 **游戏循环** 和 **游戏时钟**
- 理解 **精灵** 和 **精灵组**

项目准备

- 新建 **飞机大战** 项目
- 新建一个 `wd_01_pygame` 入门.py
- 导入 **游戏素材图片**

游戏的第一印象

1. 把一些 **静止的图像** 绘制到 **游戏窗口** 中
2. 根据 **用户的交互** 或其他情况，**移动** 这些图像，产生动画效果
3. 根据 **图像之间** 是否发生重叠，判断 **敌机是否被摧毁** 等其他情况

01. 使用 `pygame` 创建图形窗口

小节目标

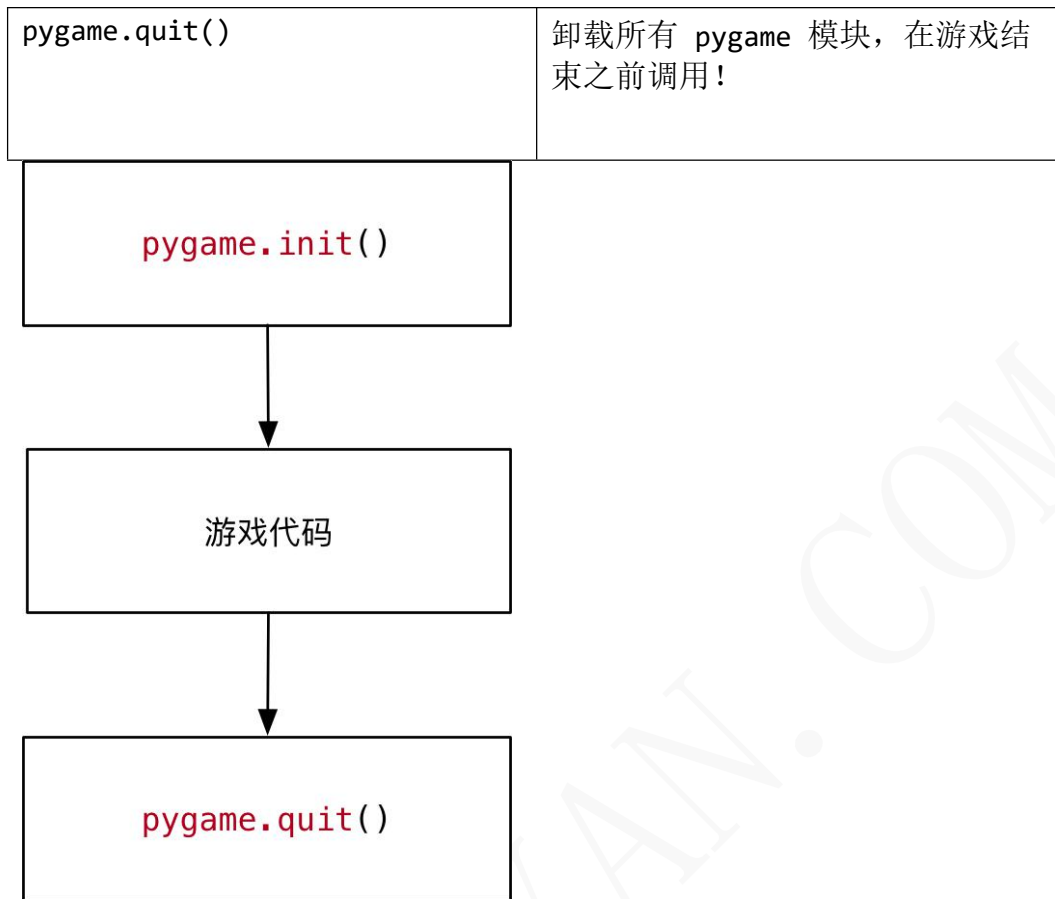
- 游戏的初始化和退出
- 理解游戏中的坐标系
- 创建游戏主窗口
- 简单的游戏循环

可以将图片素材 **绘制** 到 **游戏的窗口** 上，开发游戏之前需要先知道 **如何建立游戏窗口**！

1.1 游戏的初始化和退出

1. 要使用 `pygame` 提供的所有功能之前，需要调用 `init` 方法
2. 在游戏结束前需要调用一下 `quit` 方法

方法	说明
<code>pygame.init()</code>	导入并初始化所有 <code>pygame</code> 模块，使用其他模块之前，必须先调用 <code>init</code> 方法



001_pygame 的 `init` 和 `quit`

```
import pygame
```

```
pygame.init()
```

```
# 编写游戏的代码
```

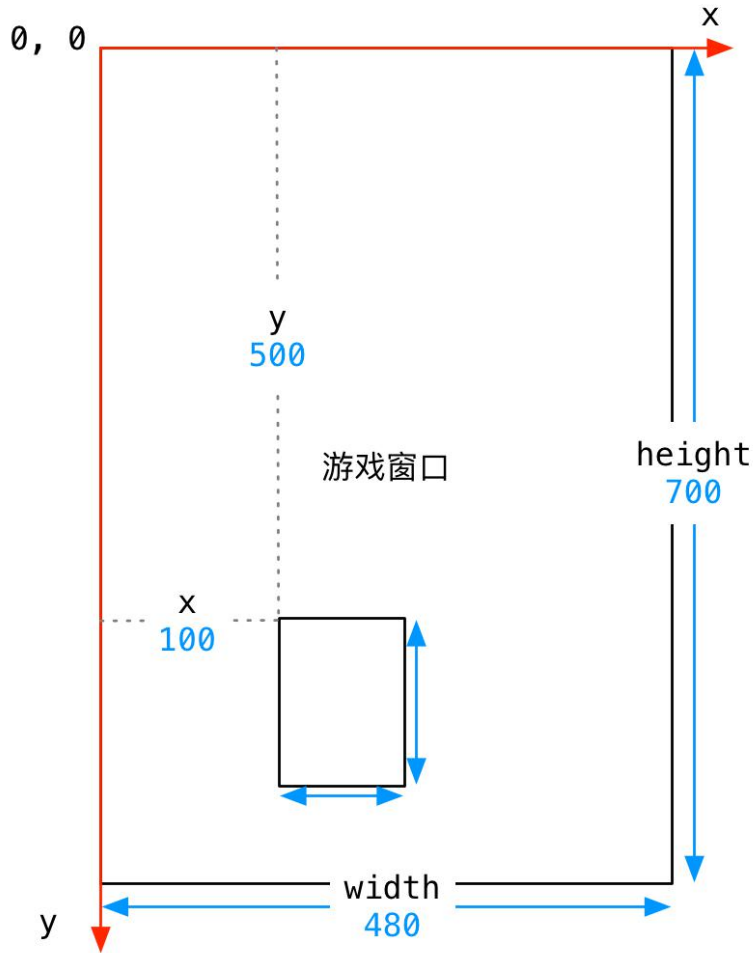
```
print("游戏的代码...")
```

```
pygame.quit()
```

1.2 理解游戏中的坐标系

3. 坐标系

1. 原点 在 左上角 $(0, 0)$
2. **x** 轴 水平方向向 右，逐渐增加
3. **y** 轴 垂直方向向 下，逐渐增加



002_游戏窗口和坐标系

1. 在游戏中，所有可见的元素 都是以 矩形区域 来描述位置的
要描述一个矩形区域有四个要素：(x, y) (width, height)
2. pygame 专门提供了一个类 `pygame.Rect` 用于描述 矩形区域
`Rect(x, y, width, height) -> Rect`

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

003_pygame.Rect

提示

3. `pygame.Rect` 是一个比较特殊的类，内部只是封装了一些数字计算

4. 不执行 `pygame.init()` 方法同样能够直接使用

案例演练

需求

- 定义 `hero_rect` 矩形描述 英雄的位置和大小
- 输出英雄的 坐标原点 (x 和 y)
- 输出英雄的 尺寸 (宽度 和 高度)

```
import pygame
```

```
hero_rect = pygame.Rect(100, 500, 120, 125)
```

```
print("英雄的原点 %d %d" % (hero_rect.x, hero_rect.y))
print("英雄的尺寸 %d %d" % (hero_rect.width, hero_rect.height))
# size 属性会返回矩形区域的 (宽, 高) 元组
print("%d %d" % hero_rect.size)
```

1.3 创建游戏主窗口

5. `pygame` 专门提供了一个 模块 `pygame.display` 用于创建、管理 游戏窗口

方法	说明
<code>pygame.display.set_mode()</code>	初始化游戏显示窗口
<code>pygame.display.update()</code>	刷新屏幕内容显示，稍后使用

`set_mode` 方法

```
set_mode(resolution=(0,0), flags=0, depth=0) -> Surface
```

1. 作用 —— 创建游戏显示窗口

2. 参数

1. `resolution` 指定屏幕的 宽 和 高，默认创建的窗口大小和屏幕大小一致
2. `flags` 参数指定屏幕的附加选项，例如是否全屏等等，默认不需要传递
3. `depth` 参数表示颜色的位数，默认自动匹配

3. 返回值

1. **暂时** 可以理解为 **游戏的屏幕**，**游戏的元素** 都需要被绘制到 **游戏的屏幕** 上
4. **注意**：必须使用变量记录 `set_mode` 方法的返回结果！因为：后续所有的图像绘制都基于这个返回结果

创建游戏主窗口

```
screen = pygame.display.set_mode((480, 700))
```

1.4 简单的游戏循环

1. 为了做到游戏程序启动后，**不会立即退出**，通常会在游戏程序中增加一个 **游戏循环**
2. 所谓 **游戏循环** 就是一个 **无限循环**
3. 在 **创建游戏窗口** 代码下方，增加一个无限循环

注意：游戏窗口不需要重复创建

```
import pygame
import time
pygame.init()
```

创建游戏的窗口 480 * 700

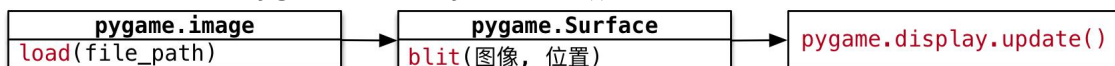
```
screen = pygame.display.set_mode((480, 700))
```

```
while True:
    time.sleep(1)
```

```
pygame.quit()
```

02. 理解 图像 并实现图像绘制

1. 在游戏中，能够看到的 **游戏元素** 大多都是 **图像**
图像文件 初始是保存在磁盘上的，如果需要使用，**第一步** 就需要 **被加载到内存**
2. 要在屏幕上 **看到某一个图像的内容**，需要按照三个步骤：
 - 使用 `pygame.image.load()` 加载图像的数据
 - 使用 **游戏屏幕** 对象，调用 `blit` 方法 将图像绘制到指定位置
 - 调用 `pygame.display.update()` 方法更新整个屏幕的显示



004_加载和显示图像

提示：要想在屏幕上看到绘制的结果，就一定要调用 `pygame.display.update()` 方法

代码演练 I —— 绘制背景图像

需求

- 加载 background.png 创建背景
- 将 背景 绘制在屏幕的 (0, 0) 位置
- 调用屏幕更新显示背景图像

```
import pygame

pygame.init()

# 创建游戏的窗口 480 * 700
screen = pygame.display.set_mode((480, 700))

# 绘制背景图像
# 1> 加载图像数据
bg = pygame.image.load("./images/background.png")
# 2> blit 绘制图像
screen.blit(bg, (0, 0))
# 3> update 更新屏幕显示
pygame.display.update()

while True:
    pass

pygame.quit()
```

代码演练 II —— 绘制英雄图像

需求

- 加载 me1.png 创建英雄飞机
- 将 英雄飞机 绘制在屏幕的 (200, 500) 位置
- 调用屏幕更新显示飞机图像

在代码演练 I 中增加如下代码：

```
# 1> 加载图像
hero = pygame.image.load("./images/me1.png")

# 2> 绘制在屏幕
screen.blit(hero, (200, 500))

# 3> 更新显示
pygame.display.update()
```

透明图像

1. png 格式的图像是支持 透明 的
2. 在绘制图像时，透明区域 不会显示任何内容
3. 但是如果下方已经有内容，会 透过 透明区域 显示出来

理解 update() 方法的作用

可以在 screen 对象完成 所有 blit 方法之后，统一调用一次 display.update 方法，同样可以在屏幕上 看到最终的绘制结果

1. 使用 display.set_mode() 创建的 screen 对象 是一个 内存中的屏幕数据对象

可以理解成是 油画 的 画布

2. screen.blit 方法可以在 画布 上绘制很多 图像
 1. 例如：英雄、敌机、子弹...
 2. 这些图像 有可能 会彼此 重叠或者覆盖
3. display.update() 会将 画布 的 最终结果 绘制在屏幕上，这样可以 提高 屏幕绘制效率，增加游戏的流畅度

案例调整

绘制背景图像

1> 加载图像

```
bg = pygame.image.load("./images/background.png")
```

2> 绘制在屏幕

```
screen.blit(bg, (0, 0))
```

绘制英雄图像

1> 加载图像

```
hero = pygame.image.load("./images/me1.png")
```

2> 绘制在屏幕

```
screen.blit(hero, (200, 500))
```

3> 更新显示 - update 方法会把之前所有绘制的结果，一次性更新到屏幕窗口上

```
pygame.display.update()
```

上面的意义在于我们不用每次放置一张图都去 update

03. 理解 游戏循环 和 游戏时钟

现在 英雄飞机 已经被绘制到屏幕上了，怎么能够让飞机移动呢？

3.1 游戏中的动画实现原理

1. 跟 **电影** 的原理类似，游戏中的动画效果，本质上是 **快速** 的在屏幕上绘制 **图像**

电影是将多张 **静止的电影胶片** **连续、快速**的播放，产生连贯的视觉效果！

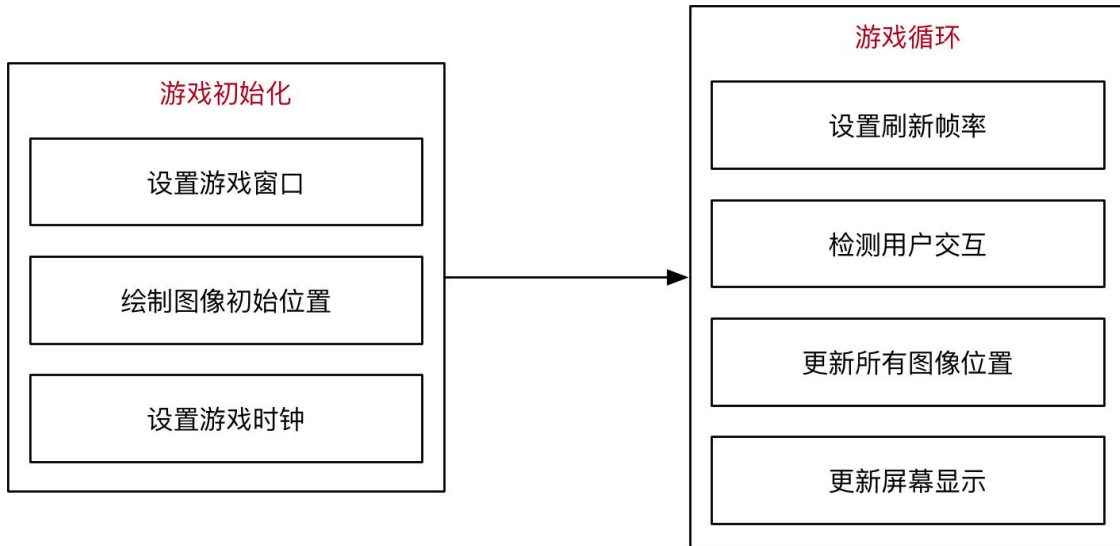
2. 一般在电脑上 **每秒绘制 60 次**，就能够达到非常 **连续 高品质** 的动画效果
 1. 每次绘制的结果被称为 **帧 Frame**



3.2 游戏循环

游戏的两个组成部分

游戏循环的开始 就意味着 游戏的正式开始



005_游戏主模块

游戏循环的作用

- 保证游戏 不会直接退出
- 变化图像位置 —— 动画效果
 1. 每隔 1 / 60 秒 移动一下所有图像的位置
 2. 调用 `pygame.display.update()` 更新屏幕显示
- 检测用户交互 —— 按键、鼠标等...

3.3 游戏时钟

1. `pygame` 专门提供了一个类 `pygame.time.Clock` 可以非常方便的设置屏幕绘制速度 —— 刷新帧率
2. 要使用 时钟对象 需要两步：
 1. 在 游戏初始化 创建一个 时钟对象
 2. 在 游戏循环 中让时钟对象调用 `tick(帧率)` 方法
3. `tick` 方法会根据 上次被调用的时间，自动设置 游戏循环 中的延时

3. 创建游戏时钟对象

```
clock = pygame.time.Clock()
i = 0
```

游戏循环

```
while True:
```

```
    # 设置屏幕刷新帧率，每秒 60 次
    clock.tick(60)
```

```
    print(i)
    i += 1
```

3.4 英雄的简单动画实现

需求

- 在 **游戏初始化** 定义一个 `pygame.Rect` 的变量记录英雄的初始位置
- 在 **游戏循环** 中每次让 **英雄** 的 `y - 1` —— 向上移动
- `y <= 0` 将英雄移动到屏幕的底部

提示：

1. 每一次调用 `update()` 方法之前，需要把 **所有的游戏图像都重新绘制一遍**
2. 而且应该 **最先** 重新绘制 **背景图像**

实现如下：

```
import pygame

# 游戏的初始化
pygame.init()

# 创建游戏的窗口 480 * 700
screen = pygame.display.set_mode((480, 700))

# 绘制背景图像
bg = pygame.image.load("./images/background.png")
screen.blit(bg, (0, 0))
# pygame.display.update()

# 绘制英雄的飞机
hero = pygame.image.load("./images/me1.png")
screen.blit(hero, (150, 300))

# 可以在所有绘制工作完成之后，统一调用 update 方法
pygame.display.update()

# 创建时钟对象
clock = pygame.time.Clock()

# 1. 定义 rect 记录飞机的初始位置
hero_rect = pygame.Rect(150, 300, 102, 126)

# 游戏循环 -> 意味着游戏的正式开始！
while True:
```

```
# 可以指定循环体内部的代码执行的频率
clock.tick(60)

# 2. 修改飞机的位置
hero_rect.y -= 1

# 3. 调用 blit 方法绘制图像
screen.blit(bg, (0, 0))
screen.blit(hero, hero_rect)

# 4. 调用 update 方法更新显示
pygame.display.update()

pygame.quit()
```

如果飞机要循环飞行

```
# 4. 定义英雄的初始位置
hero_rect = pygame.Rect(150, 500, 102, 126)

while True:

    # 可以指定循环体内部的代码执行的频率
    clock.tick(60)

    # 更新英雄位置
    hero_rect.y -= 1

    # 如果移出屏幕, 则将英雄的顶部移动到屏幕底部
    if hero_rect.y <= 0:
        hero_rect.y = 700

    # 绘制背景图片
    screen.blit(bg, (0, 0))
    # 绘制英雄图像
    screen.blit(hero, hero_rect)

    # 更新显示
    pygame.display.update()
```

作业

- 英雄向上飞行, 当 英雄完全从上方飞出屏幕后
- 将飞机移动到屏幕的底部

```
if hero_rect.y + hero_rect.height <= 0:
    hero_rect.y = 700
```

提示

Rect 的属性 `bottom = y + height`

```
if hero_rect.bottom <= 0:
    hero_rect.y = 700
```

3.5 在游戏循环中 监听 事件

事件 event

1. 就是游戏启动后，用户针对游戏所做的操作
2. 例如：点击关闭按钮，点击鼠标，按下键盘...

监听

3. 在 游戏循环 中，判断用户 具体的操作

只有 捕获 到用户具体的操作，才能有针对性的做出响应

代码实现

1. pygame 中通过 `pygame.event.get()` 可以获得 用户当前所做动作 的 事件列表

用户可以同一时间做很多事情

2. 提示：这段代码非常的固定，几乎所有的 pygame 游戏都 大同小异！

```
import pygame
```

```
# 游戏的初始化
pygame.init()
```

```
# 创建游戏的窗口 480 * 700
screen = pygame.display.set_mode((480, 700))
```

```
# 绘制背景图像
bg = pygame.image.load("./images/background.png")
screen.blit(bg, (0, 0))
# pygame.display.update()
```

```
# 绘制英雄的飞机
hero = pygame.image.load("./images/me1.png")
screen.blit(hero, (150, 300))
```

```
# 可以在所有绘制工作完成之后，统一调用 update 方法
pygame.display.update()
```

```
# 创建时钟对象
clock = pygame.time.Clock()
```

```
# 1. 定义 rect 记录飞机的初始位置
hero_rect = pygame.Rect(150, 300, 102, 126)

# 游戏循环 -> 意味着游戏的正式开始!
while True:

    # 可以指定循环体内部的代码执行的频率
    clock.tick(60)

    # 捕获事件
    event_list = pygame.event.get()
    if len(event_list) > 0:
        print(event_list)

    # 2. 修改飞机的位置
    hero_rect.y -= 1

    # 判断飞机的位置
    if hero_rect.y <= 0:
        hero_rect.y = 700

    # 3. 调用 blit 方法绘制图像
    screen.blit(bg, (0, 0))
    screen.blit(hero, hero_rect)

    # 4. 调用 update 方法更新显示
    pygame.display.update()

pygame.quit()
```

针对退出，我们可以在 while 循环中增加如下代码：

```
for event in pygame.event.get():

    # 判断事件类型是否是退出事件
    if event.type == pygame.QUIT:
        print("游戏退出...")

    # quit 卸载所有的模块
    pygame.quit()

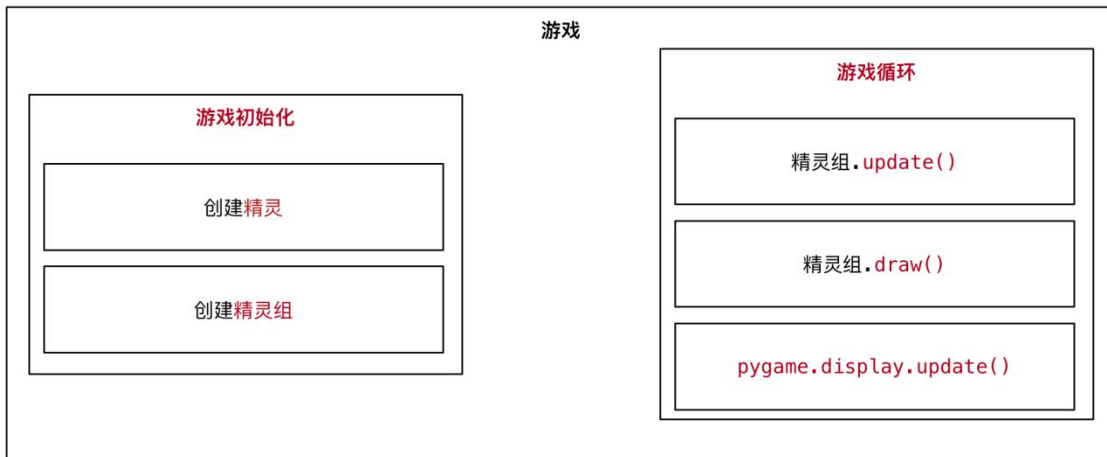
    # exit() 直接终止当前正在执行的程序
    exit()
```

04. 理解 精灵 和 精灵组

4.1 精灵 和 精灵组

1. 在刚刚完成的案例中，**图像加载、位置变化、绘制图像** 都需要程序员编写代码分别处理
2. 为了简化开发步骤，pygame 提供了两个类
 1. pygame.sprite.Sprite —— 存储 **图像数据 image** 和 **位置 rect** 的对象
 2. pygame.sprite.Group

精灵 (需要派生子类)	精灵组
image 记录图像数据	__init__(self, *精灵):
rect 记录在屏幕上的位置	add(*sprites): 向组中增加精灵
update(*args): 更新精灵位置	sprites(): 返回所有精灵列表
kill(): 从所有组中删除	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



006_pygame.Sprite

精灵

1. 在游戏开发中，通常把 **显示图像的对象** 叫做精灵 Sprite
2. **精灵** 需要 有 **两个重要的属性**
 1. **image** 要显示的图像
 2. **rect** 图像要显示在屏幕的位置
3. 默认的 **update()** 方法什么事情也没做
 1. 子类可以重写此方法，在每次刷新屏幕时，更新精灵位置
4. **注意:** pygame.sprite.Sprite 并没有提供 **image** 和 **rect** 两个属性
 1. 需要程序员从 **pygame.sprite.Sprite** 派生子类
 2. 并在 **子类** 的 **初始化方法** 中，设置 **image** 和 **rect** 属性

精灵组

1. 一个 精灵组 可以包含多个 精灵 对象
 2. 调用 精灵组 对象的 `update()` 方法
 1. 可以 自动 调用 组内每一个精灵 的 `update()` 方法
 3. 调用 精灵组 对象的 `draw(屏幕对象)` 方法
 1. 可以将 组内每一个精灵 的 `image` 绘制在 `rect` 位置
- `Group(*sprites) -> Group`

注意：仍然需要调用 `pygame.display.update()` 才能在屏幕看到最终结果

4.2 派生精灵子类

- 新建 `plane_sprites.py` 文件
- 定义 `GameSprite` 继承自 `pygame.sprite.Sprite`

注意

1. 如果一个类的 父类 不是 `object`
2. 在重写 初始化方法 时，一定要 先 `super()` 一下父类的 `__init__` 方法
3. 保证父类中实现的 `__init__` 代码能够被正常执行

GameSprite
<code>image</code> <code>rect</code> <code>speed</code>
<code>__init__(self, image_name, speed=1):</code> <code>update(self):</code>

007_GameSprite

属性

1. `image` 精灵图像，使用 `image_name` 加载
2. `rect` 精灵大小，默认使用图像大小
3. `speed` 精灵移动速度，默认为 1

方法

4. `update` 每次更新屏幕时在游戏循环内调用
 1. 让精灵的 `self.rect.y += self.speed`

提示

`image` 的 `get_rect()` 方法，可以返回 `pygame.Rect(0, 0, 图像宽, 图像高)` 的对象

首先我们看下代码里的精灵类


```
import pygame
```

```
class GameSprite(pygame.sprite.Sprite):
    """ 游戏精灵基类 """

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

        # 加载图像
        self.image = pygame.image.load(image_name)
        # 设置尺寸
        self.rect = self.image.get_rect()
        # 记录速度
        self.speed = speed

    def update(self, *args):
        # 默认在垂直方向移动
        self.rect.y += self.speed
```

4.3 使用 游戏精灵 和 精灵组 创建敌机

需求

1. 使用刚刚派生的 游戏精灵 和 精灵组 创建 敌机 并且实现敌机动画

步骤

- 使用 from 导入 plane_sprites 模块
 1. from 导入的模块可以 直接使用
 2. import 导入的模块需要通过 模块名. 来使用
- 在 游戏初始化 创建 精灵对象 和 精灵组对象
- 在 游戏循环中 让 精灵组 分别调用 update() 和 draw(screen) 方法

职责

2. 精灵

1. 封装 图像 image、位置 rect 和 速度 speed
2. 提供 update() 方法，根据游戏需求，更新位置 rect

3. 精灵组

1. 包含 多个 精灵对象
2. update 方法，让精灵组中的所有精灵调用 update 方法更新位置

3. `draw(screen)` 方法，在 `screen` 上绘制精灵组中的所有精灵

3 游戏框架搭建

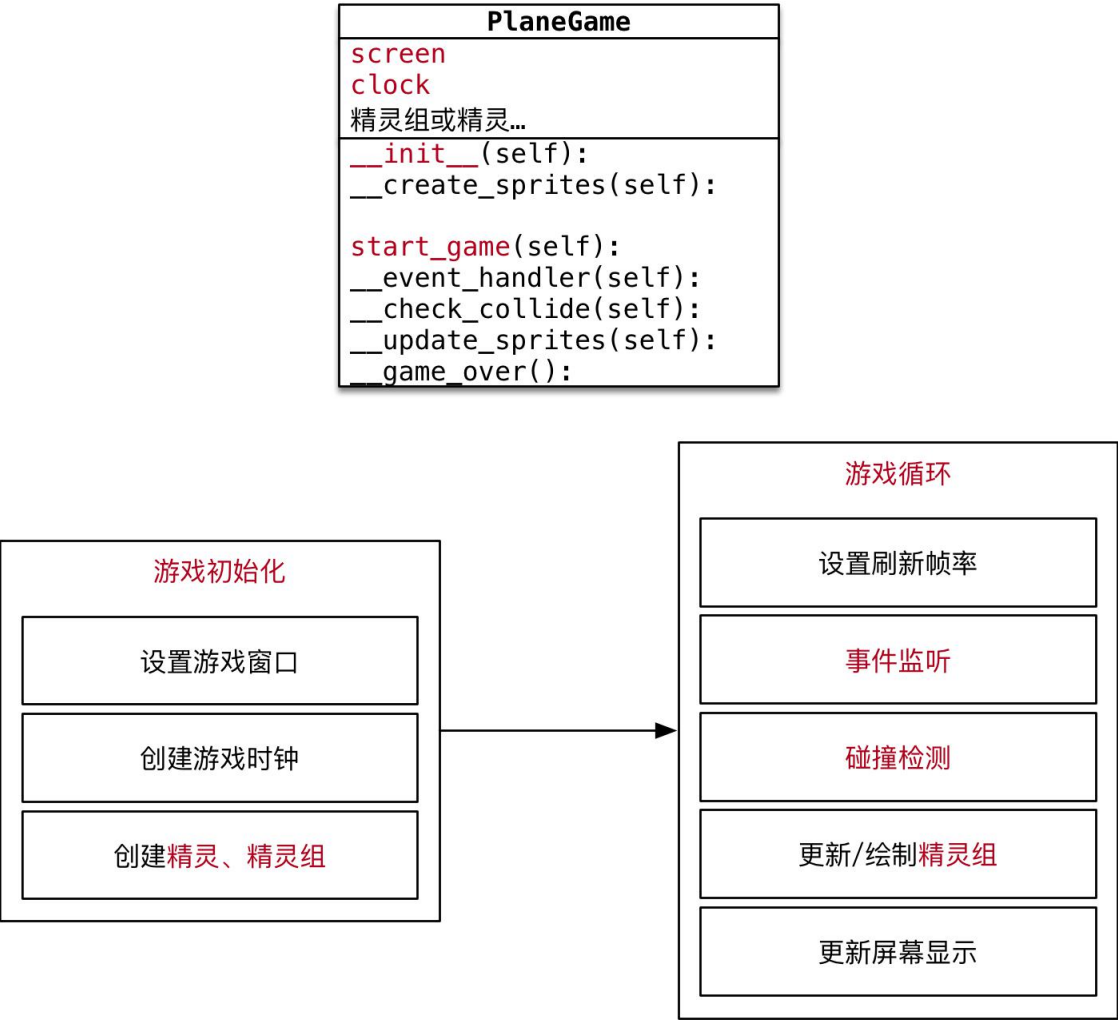
目标 —— 使用 面相对象 设计 飞机大战游戏类

目标

- 明确主程序职责
- 实现主程序类
- 准备游戏精灵组

01. 明确主程序职责

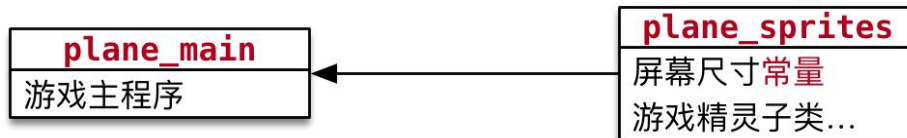
- 回顾 快速入门案例，一个游戏主程序的 职责 可以分为两个部分：
 - 游戏初始化
 - 游戏循环
- 根据明确的职责，设计 `PlaneGame` 类如下：



<code>__check_collide(self)</code>	碰撞检测 —— 子弹销毁敌机、敌机撞毁英雄
<code>__update_sprites(self)</code>	精灵组更新和绘制
<code>__game_over()</code>	游戏结束

02. 实现飞机大战主游戏类

2.1 明确文件职责



011_程序文件职责

- `plane_main`
 1. 封装 主游戏类
 2. 创建 游戏对象
 3. 启动游戏
- `plane_sprites`
 - 封装游戏中 所有 需要使用的 精灵子类
 - 提供游戏的 相关工具

代码实现

- 新建 `plane_main.py` 文件，并且设置为可执行
- 编写 基础代码

```
import pygame
from plane_sprites import *
```

```
class PlaneGame(object):
    """ 飞机大战主游戏 """

    def __init__(self):
        print("游戏初始化")

    def start_game(self):
```

```
print("开始游戏...")
```

```
if __name__ == '__main__':
    # 创建游戏对象
    game = PlaneGame()

    # 开始游戏
    game.start_game()
```

2.3 游戏初始化部分

- 完成 `__init__()` 代码如下：

```
def __init__(self):
    print("游戏初始化")

    # 1. 创建游戏的窗口
    self.screen = pygame.display.set_mode((480, 700))
    # 2. 创建游戏的时钟
    self.clock = pygame.time.Clock()
    # 3. 调用私有方法，精灵和精灵组的创建
    self.__create_sprites()

def __create_sprites(self):
    pass
```

使用 常量 代替固定的数值

- 常量 —— 不变化的量
- 变量 —— 可以变化的量

应用场景

- 在开发时，可能会需要使用 **固定的数值**，例如 **屏幕的高度** 是 700
- 这个时候，建议 **不要** 直接使用固定数值，而应该使用 **常量**
- 在开发时，为了保证代码的可维护性，尽量不要使用 **魔法数字**

常量的定义

- 定义 **常量** 和 定义 **变量** 的语法完全一样，都是使用 **赋值语句**
- **常量** 的 **命名** 应该 **所有字母都使用大写**，单词与单词之间使用下划线连接

常量的好处

- 阅读代码时，通过 **常量名** 见名之意，不需要猜测数字的含义
- 如果需要 **调整值**，只需要 **修改常量定义** 就可以实现 **统一修改**

提示：Python 中并没有真正意义的常量，只是通过命名的约定 —— 所有字母都是大写的就是常量，开发时不要轻易的修改！

代码调整

- 在 plane_sprites.py 中增加常量定义
import pygame

游戏屏幕大小

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

- 修改 plane_main.py 中的窗口大小
self.screen = pygame.display.set_mode(SCREEN_RECT.size)

2.4 游戏循环部分

- 完成 start_game() 基础代码如下:

```
def start_game(self):
    """开始游戏"""

    print("开始游戏...")

    while True:

        # 1. 设置刷新帧率
        self.clock.tick(60)

        # 2. 事件监听
        self.__event_handler()

        # 3. 碰撞检测
        self.__check_collide()

        # 4. 更新精灵组
        self.__update_sprites()

        # 5. 更新屏幕显示
        pygame.display.update()

    def __event_handler(self):
        """事件监听"""

        for event in pygame.event.get():

            if event.type == pygame.QUIT:
                PlaneGame.__game_over()

    def __check_collide(self):
        """碰撞检测"""
        pass
```

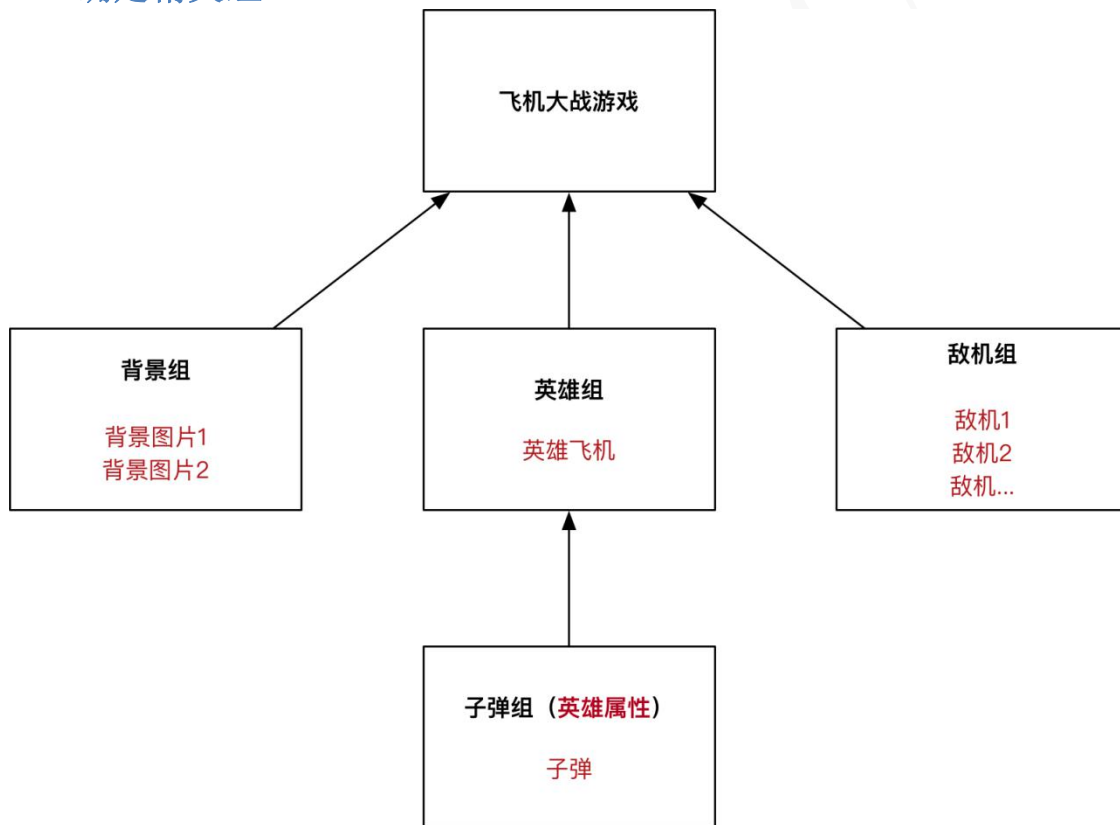
```
def __update_sprites(self):
    """更新精灵组"""
    pass

@staticmethod
def __game_over():
    """游戏结束"""

    print("游戏结束")
    pygame.quit()
    exit()
```

03. 准备游戏精灵组

3.1 确定精灵组



010_精灵组确定

3.2 代码实现

- 创建精灵组方法

```
def __create_sprites(self):
    """创建精灵组"""
```

```
# 背景组
self.back_group = pygame.sprite.Group()
# 敌机组
self.enemy_group = pygame.sprite.Group()
# 英雄组
self.hero_group = pygame.sprite.Group()
```

- 更新精灵组方法

```
def __update_sprites(self):
    """更新精灵组"""

    for group in [self.back_group, self.enemy_group, self.hero_group]:

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


4 游戏背景

目标

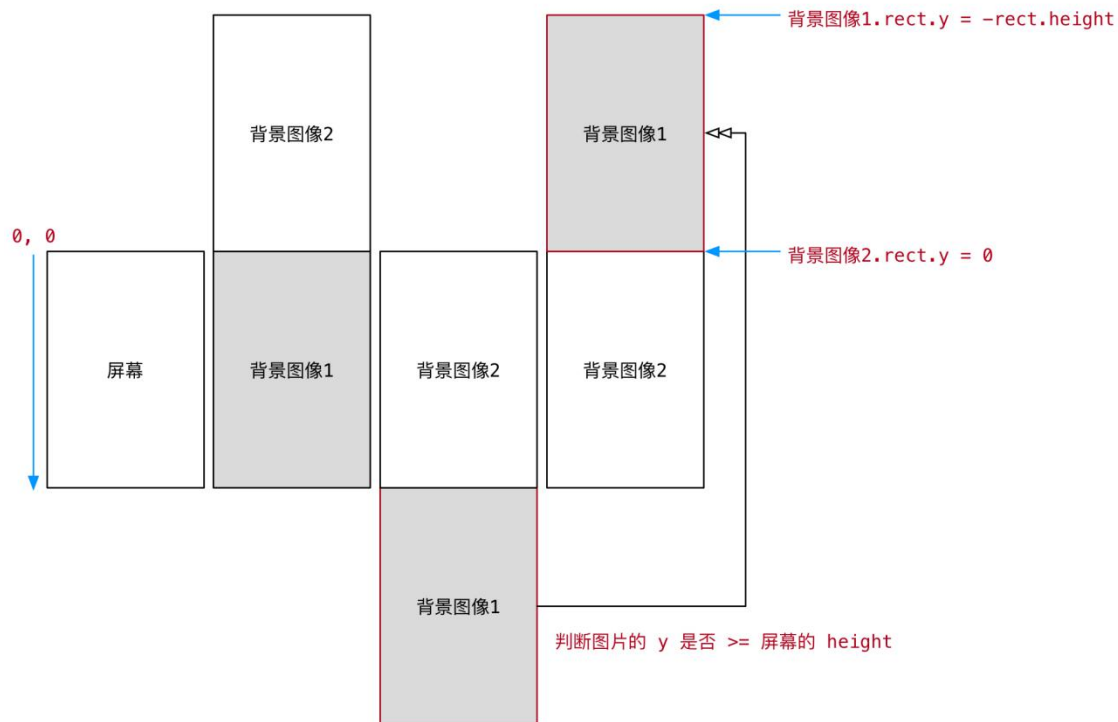
- 背景交替滚动的思路确定
- 显示游戏背景

01. 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

1.1 实现思路分析

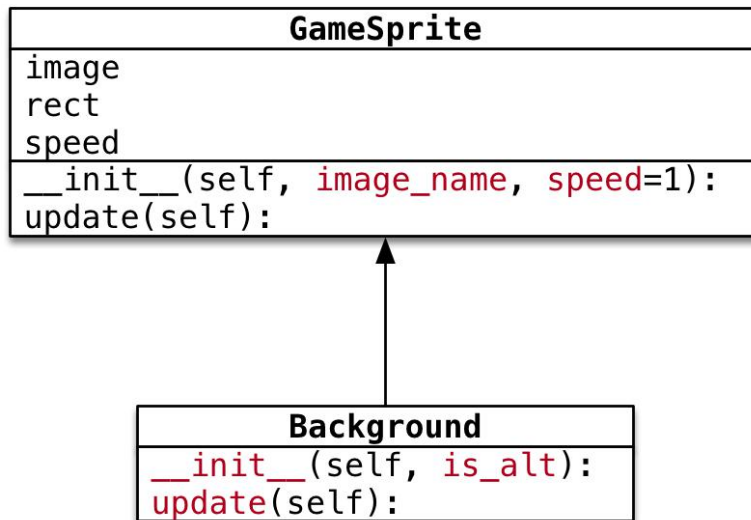


013_背景图片交替滚动

解决办法

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`

1.2 设计背景类



012_派生 Background 子类-

- 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

02. 显示游戏背景

2.1 背景精灵的基本实现

- 在 plane_sprites 新建 Background 继承自 GameSprite

```
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，将图像设置到屏幕的上方
```

```

if self.rect.y >= SCREEN_RECT.height:
    self.rect.y = -self.rect.height

```

2.2 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)

```

2.3 利用初始化方法，简化背景精灵创建

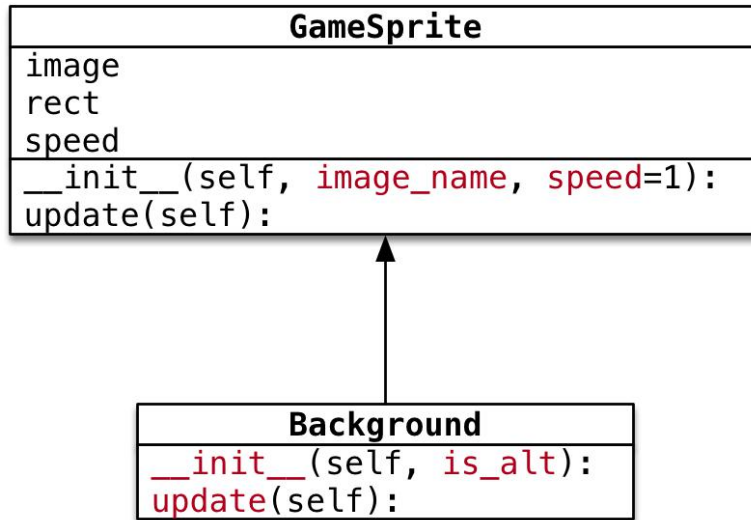
思考 —— 上一小结完成的代码存在什么样的问题？能否简化？

- 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



012_派生 Background 子类

- 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)
  
```

5 敌机出场

目标

- 使用 定时器 添加敌机
- 设计 Enemy 类

01. 使用定时器添加敌机

运行 备课代码，观察 敌机的 出现规律：

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

1.1 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

1.2 定义并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：

7. 定义 定时器常量 —— `eventid`
8. 在 初始化方法 中，调用 `set_timer` 方法 设置定时器事件
9. 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)

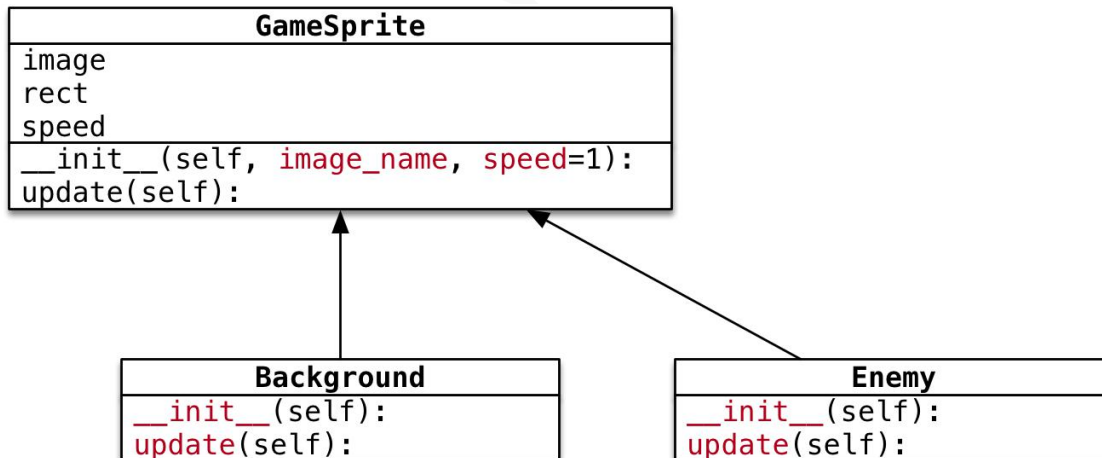
2) 监听定时器事件

- 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...")
```

02. 设计 Enemy 类

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



014_派生 Enemy 子类

- 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

2.1 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...")
```

2.2 创建敌机

演练步骤

14. 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
15. 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
16. 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

006_pygame.SpriteII

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())
```

2.3 随机敌机位置和速度

1) 导入模块

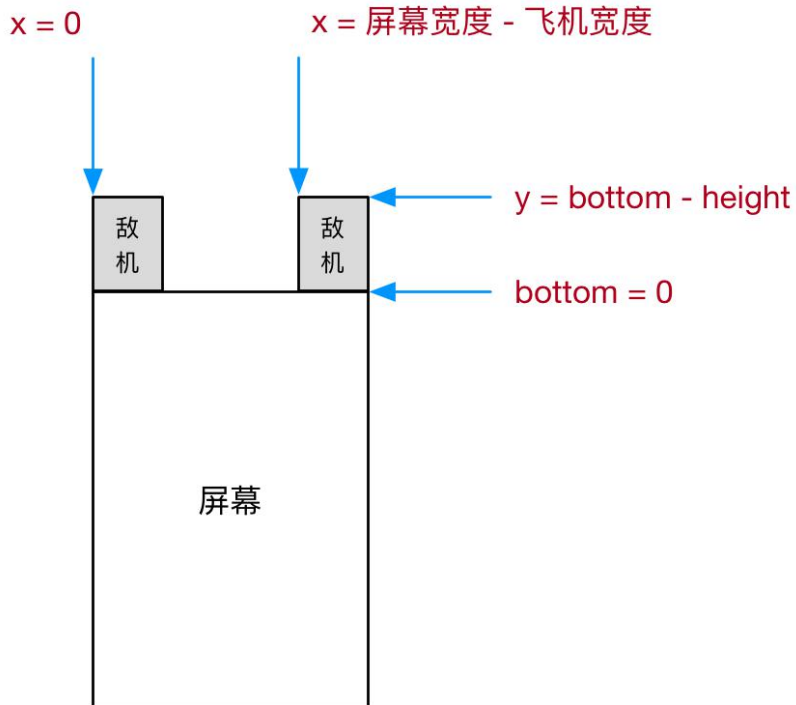
- 在导入模块时，**建议** 按照以下顺序导入

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
```

2) 随机位置



015_飞机初始位置-w360

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`

3) 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)
```

2.4 移出屏幕销毁敌机

- 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

精灵（需要派生子类）	精灵组
<code>image</code> 记录图像数据	<code>__init__(self, *精灵):</code>
<code>rect</code> 记录在屏幕上的位置	<code>add(*sprites):</code> 向组中增加精灵
<code>update(*args):</code> 更新精灵位置	<code>sprites():</code> 返回所有精灵列表
<code>kill():</code> 从所有组中删除	<code>update(*args):</code> 让组中所有精灵调用 <code>update</code> 方法
	<code>draw(Surface):</code> 将组中所有精灵的 <code>image</code> ，绘制到 <code>Surface</code> 的 <code>rect</code> 位置

006_pygame.SpriteII

- 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()
```

6 英雄登场

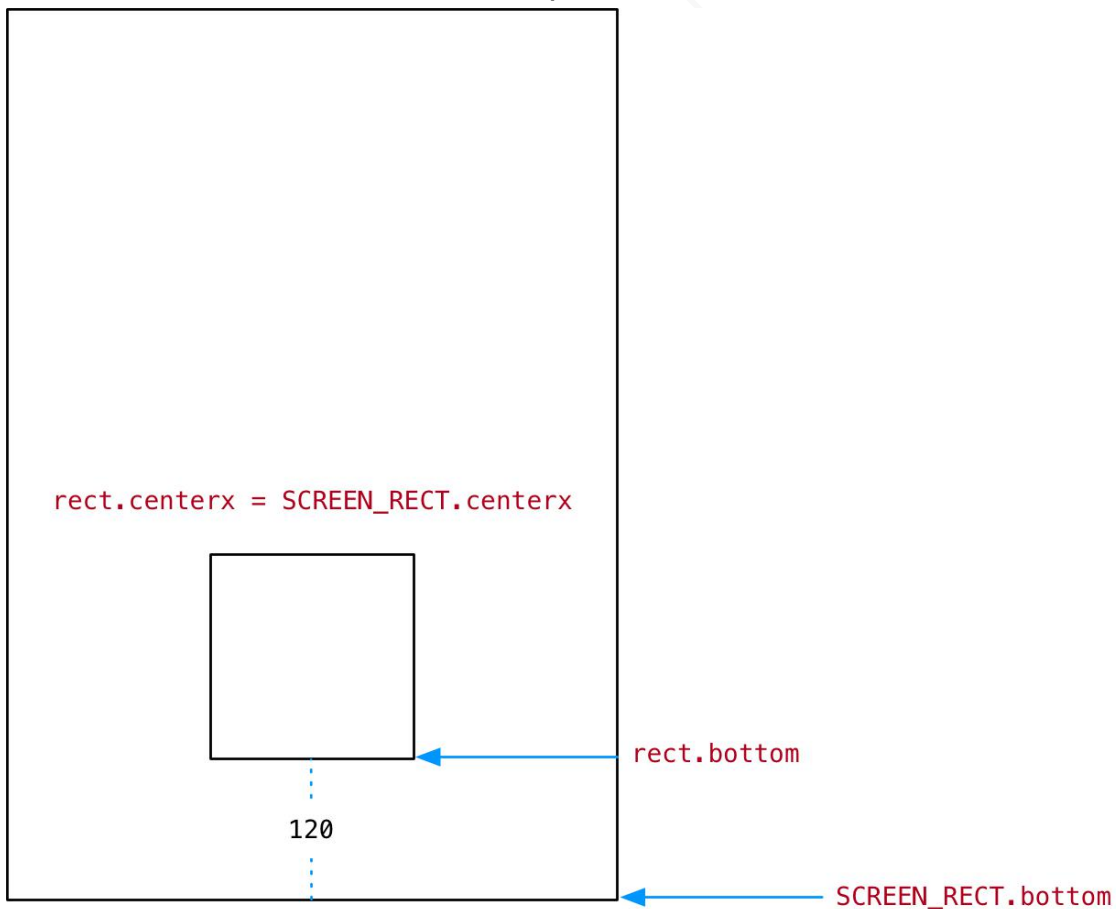
目标

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

01. 设计 英雄 和 子弹 类

英雄需求

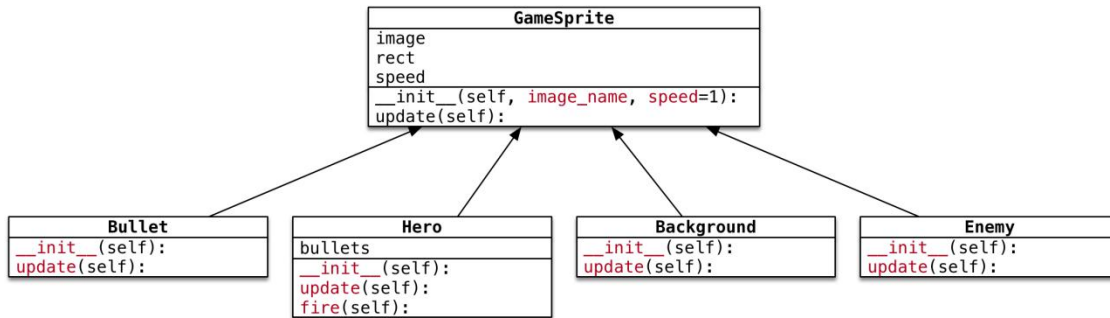
1. 游戏启动后，**英雄** 出现在屏幕的 **水平中间** 位置，距离 **屏幕底部 120 像素**
2. **英雄** 每隔 **0.5 秒** 发射一次子弹，每次 **连发三枚子弹**
3. **英雄** 默认不会移动，需要通过 **左/右** 方向键，控制 **英雄** 在水平方向移动



017_英雄位置-w480

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



016_派生英雄和子弹子类

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

02. 创建英雄

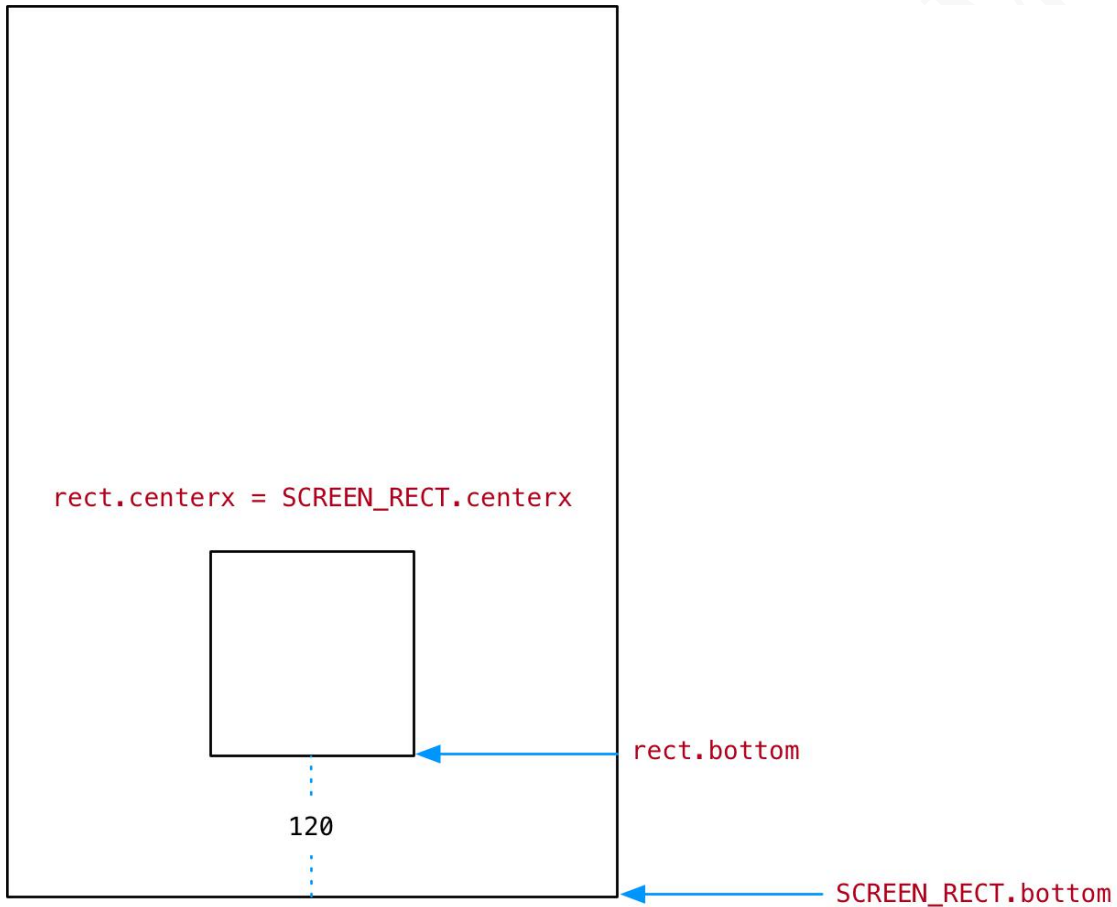
2.1 准备英雄类

- 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

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

003_pygame.Rect

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



017_英雄位置

```
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
```

2.2 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)
```

03. 移动英雄位置

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

3.1 移动英雄位置

演练步骤

3. 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
4. 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
```

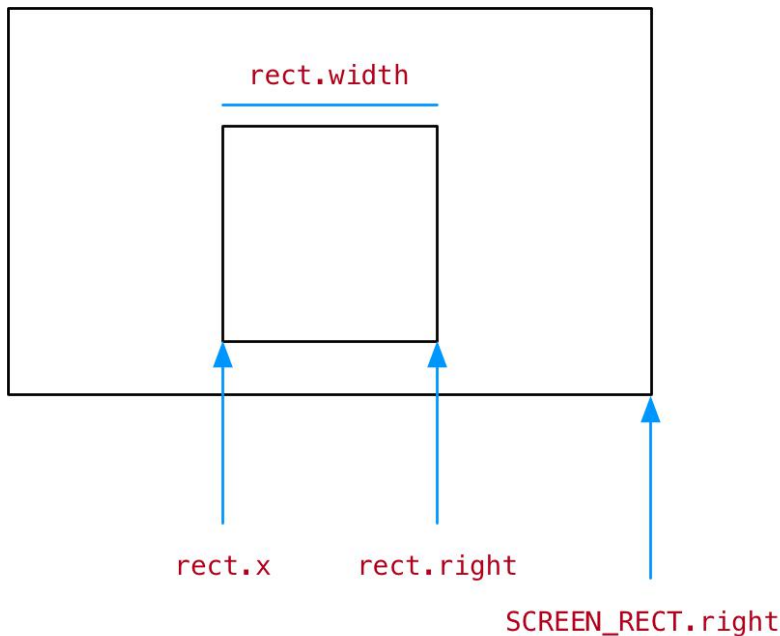
3.2 控制英雄运动边界

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

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

003_pygame.Rect

- 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



017_英雄位置 II-w408

```
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
```

04. 发射子弹

需求回顾 —— 英雄需求

5. 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
6. 英雄 每隔 0.5 秒发射一次子弹，每次 连发三枚子弹
7. 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

4.1 添加发射子弹事件

pygame 的 定时器 使用套路非常固定：

8. 定义 定时器常量 —— eventid
9. 在 初始化方法 中，调用 set_timer 方法 设置定时器事件
10. 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()
```

4.2 定义子弹类

需求回顾 —— 子弹需求

- 子弹 从 英雄 的正上方发射 沿直线 向 上方 飞行
- 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()

```

4.3 发射子弹

演练步骤

13. 在 Hero 的 初始化方法 中创建 子弹精灵组 属性
14. 修改 plane_main.py 的 __update_sprites 方法，让 子弹精灵组 调用 update 和 draw 方法
15. 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

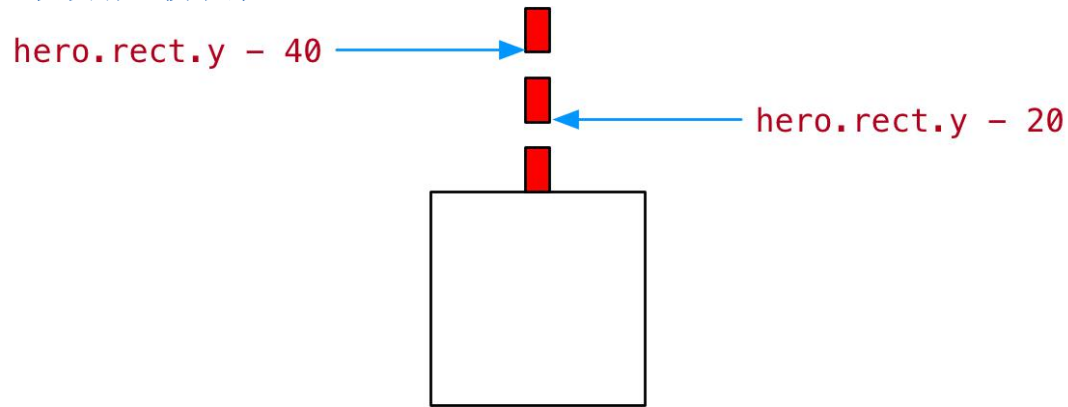
```
bullet.rect.bottom = self.rect.y - 20
```

```
bullet.rect.centerx = self.rect.centerx
```

3. 将精灵添加到精灵组

```
self.bullets.add(bullet)
```

一次发射三枚子弹



017_英雄位置 III

- 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)
```

7 碰撞检测

目标

- 了解碰撞检测方法
- 碰撞实现

01. 了解碰撞检测方法

- 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

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

02. 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, True)
```

```
    # 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()
```