[笔记][黑马 Python 之 Python 飞机大战 - 1]

Python

---

001. 项目实战-01-明确目标和实战步骤

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

---

002. 项目实战-02-确认pygame模块正确安装

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
```

验证安装

```
$ python3 -m pygame.examples.aliens
```

---

003. 快速体验-01-项目准备及游戏的第一印象

项目准备

1. 新建 **飞机大战** 项目
2. 新建一个 `hm_01_pygame入门.py`
3. 导入 **游戏素材图片**

游戏的第一印象

- 把一些 **静止的图像** 绘制到 **游戏窗口** 中
- 根据 **用户的交互** 或其他情况，**移动** 这些图像，产生动画效果
- 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

---

004. 游戏窗口-01-游戏的初始化和退出

使用 `pygame` 创建图形窗口

小节目标

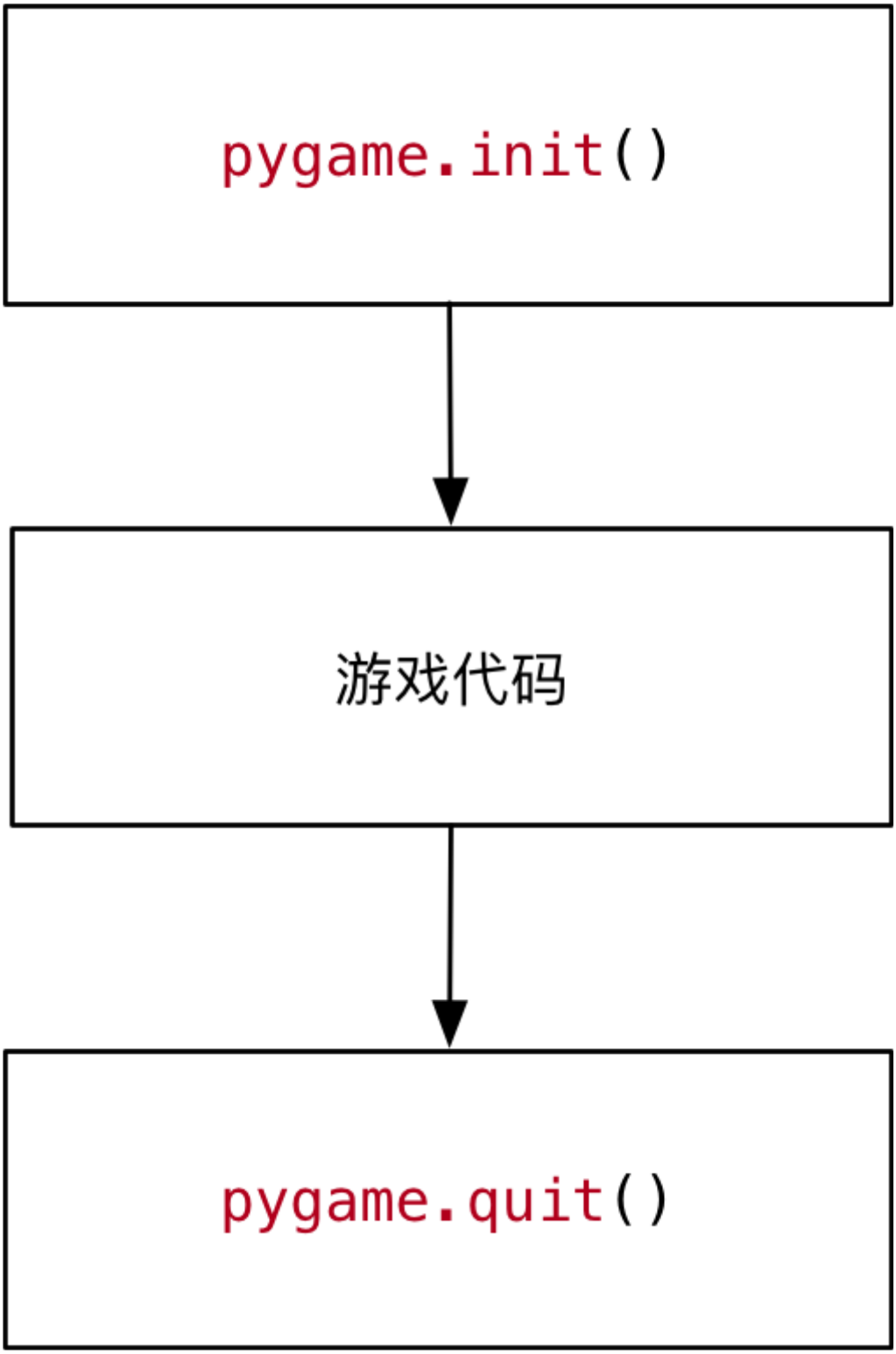
1. 游戏的初始化和退出
2. 理解游戏中的坐标系
3. 创建游戏主窗口
4. 简单的游戏循环

> 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**！

游戏的初始化和退出

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

方法	说明
`pygame.init()`	导入并初始化所有 `pygame` 模块，使用其他模块之前，必须先调用 `init` 方法
`pygame.quit()`	卸载所有 `pygame` 模块，在游戏结束之前调用！

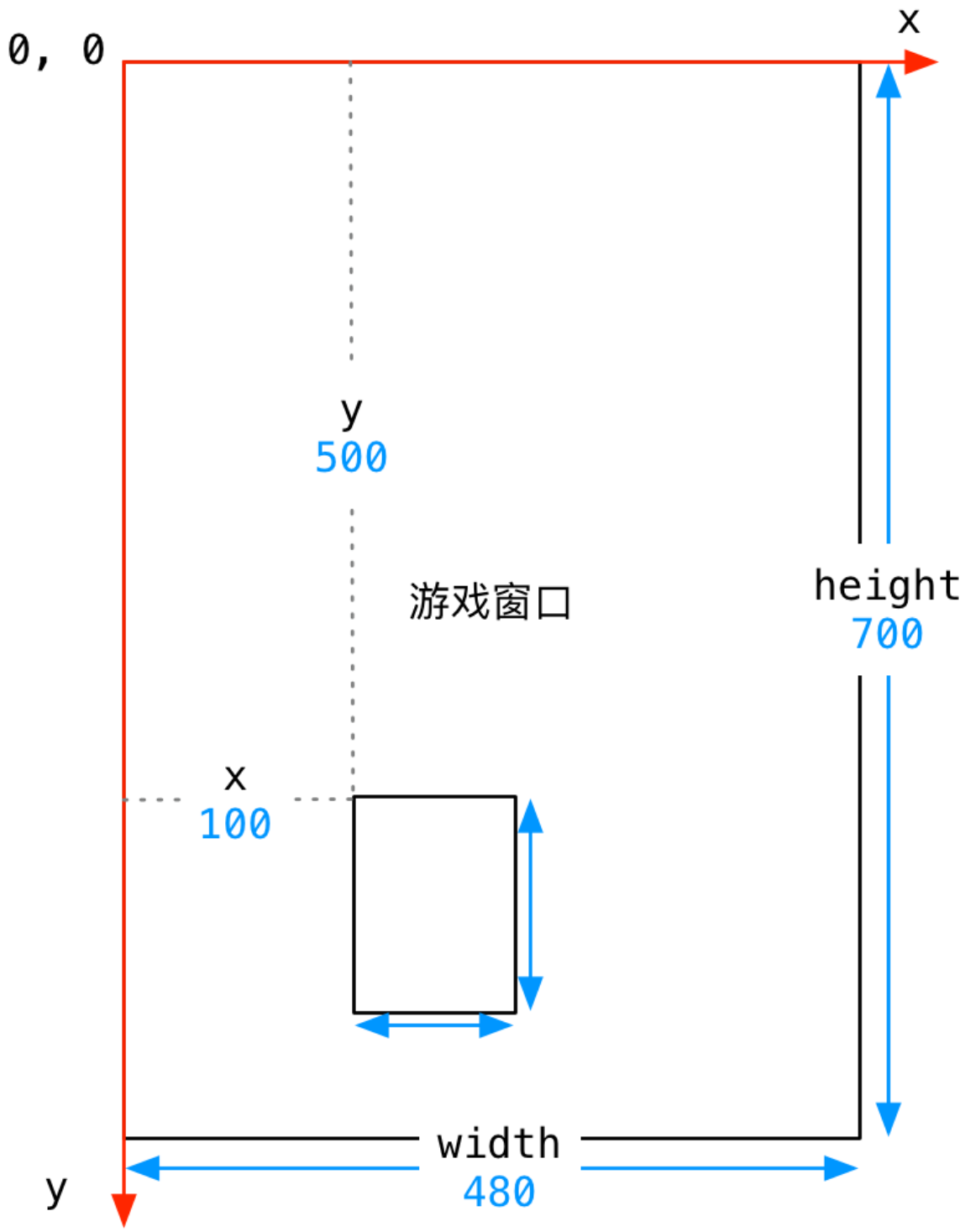



```
import pygame

pygame.init()

# 编写游戏的代码
print('游戏的代码')

pygame.quit()
```

005. 游戏窗口-02-pygame的坐标系

坐标系

- **原点** 在 **左上角** `(0, 0)`
- **x 轴** 水平方向向 **右**，逐渐增加
- **y 轴** 垂直方向向 **下**，逐渐增加

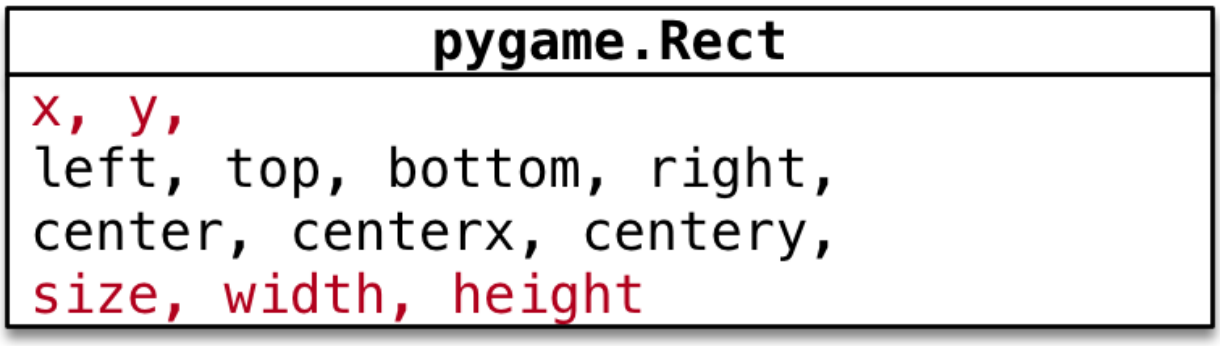


- 在游戏中，**所有可见的元素** 都是以 **矩形区域** 来描述位置的
  - 要描述一个矩形区域有四个要素：`(x, y) (width, height)`

006. 游戏窗口-03-pygame.Rect描述矩形区域

- `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
```



- `size` 属性返回一个元组，第一个值是宽，第二个值是高

提示

- `pygame.Rect` 是一个比较特殊的类，内部只是封装了一些数字计算
- 不执行 `pygame.init()` 方法同样能够直接使用

案例演练

需求

1. 定义 `hero_rect` 矩形描述 **英雄的位置和大小**
2. 输出英雄的 **坐标原点**（`x` 和 `y`）
3. 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))
print('英雄的尺寸 %d %d' % hero_rect.size)
```

---

007. 游戏窗口-04-创建游戏窗口和游戏循环

创建游戏主窗口

- `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
`pygame.display.set_mode()`	初始化游戏显示窗口
`pygame.display.update()`	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
```

- **作用** —— 创建游戏显示窗口
- **参数**
  - `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
  - `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
  - `depth` 参数表示颜色的位数，默认自动匹配
- **返回值**
  - **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
- **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))
```

简单的游戏循环

- 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**
- 所谓 **游戏循环** 就是一个 **无限循环**
- 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**

```
# 创建游戏主窗口
screen = pygame.display.set_mode((480, 700))

# 游戏循环
while True:
    pass
```

示例：

```
import pygame

pygame.init()

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

while True:
    pass

pygame.quit()
```

008. 绘制图像-01-绘制图像的三个步骤

- 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**
- 要在屏幕上 **看到某一个图像的内容**，需要按照三个步骤：
  1. 使用 `pygame.image.load()` **加载图像的数据**
  2. 使用 **游戏屏幕** 对象，调用 `blit` 方法 将图像绘制到指定位置
  3. 调用 `pygame.display.update()` 方法更新整个屏幕的显示

> 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

1. 加载 `background.png` 创建背景
2. 将 **背景** 绘制在屏幕的 `(0, 0)` 位置
3. 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> 加载图像书
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()
```

---

009. 绘制图像-02-绘制英雄和透明图像

代码演练 II —— 绘制英雄图像

需求

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

```
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> 加载图像书
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()

# 绘制英雄的飞机
hero = pygame.image.load('./images/me1.png')
screen.blit(hero, (200, 500))
pygame.display.update()
```

```
while True:
    pass

pygame.quit()
```

透明图像

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

---

010. 绘制图像-03-update方法的作用

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**

- 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**
- `screen.blit` 方法可以在 **画布** 上绘制很多 **图像**
  - 例如：**英雄**、**敌机**、**子弹**...
  - **这些图像** 有可能 会彼此 **重叠或者覆盖**
- `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()
```

---

011. 介绍计算机中的动画实现原理

> 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** ？

游戏中的动画实现原理

- 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！
- 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
  - 每次绘制的结果被称为 **帧 Frame**

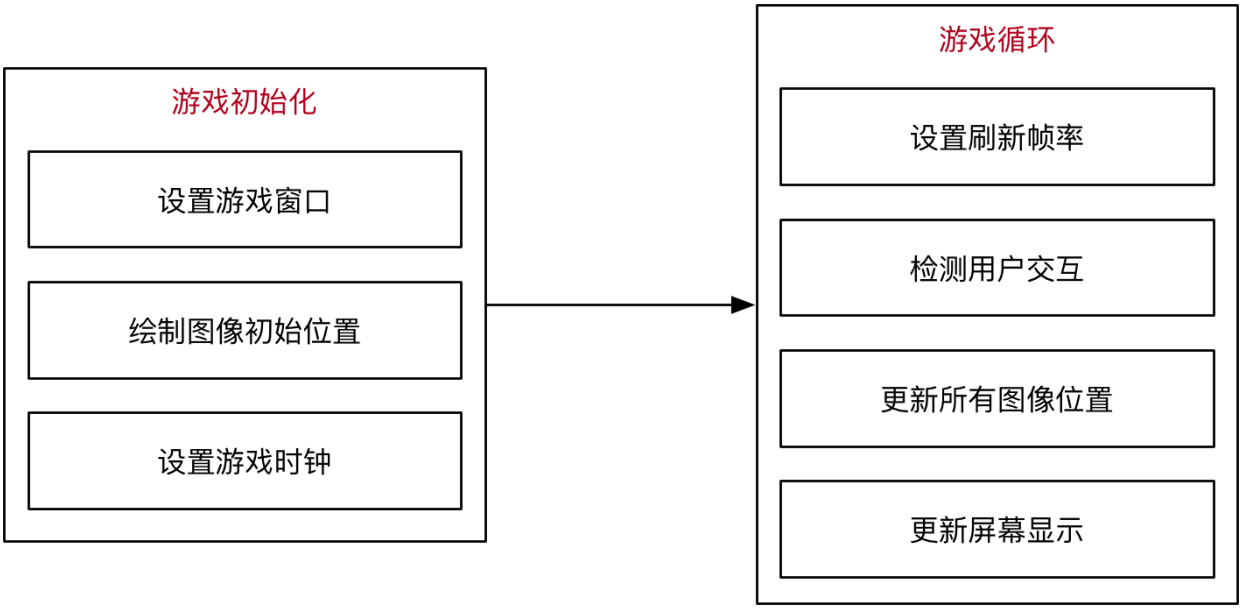
---

012. 游戏循环-01-基本概念明确下一步目标

游戏的两个组成部分

1. 游戏初始化
2. 游戏循环

> **游戏循环的开始** 就意味着 **游戏的正式开始**



游戏循环的作用

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

---

013. 游戏循环-02-利用时钟设置游戏循环的刷新帧率

- `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**
- 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
- `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

```
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
bg = pygame.image.load('./images/background.png')
screen.blit(bg, (0, 0))

# 绘制英雄的飞机
hero = pygame.image.load('./images/me1.png')
screen.blit(hero, (200, 500))

# 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()

# 游戏循环 -> 意味着游戏的正式开始！
i = 0
while True:
    # 可以指定循环体内部的代码执行的频率
    clock.tick(1)
```

```
        print(i)
        i += 1

    pygame.quit()
```

---

014. 游戏循环-03-英雄动画效果实现

需求

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

> 提示：
>
> - 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**
> - 而且应该 **最先** 重新绘制 **背景图像**
>
> 【!】`blit` 第二个参数可以是位置元组，或者矩形对象

```
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
bg = pygame.image.load('./images/background.png')
screen.blit(bg, (0, 0))

# 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()
```

---

015. 游戏循环-04-英雄循环飞行以及作业

英雄循环飞行

```
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
bg = pygame.image.load('./images/background.png')
screen.blit(bg, (0, 0))

# 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
        # 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()
```

作业

1. 英雄向上飞行，当 **英雄完全从上方飞出屏幕后**
2. 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
```

我的解答

```
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
bg = pygame.image.load('./images/background.png')
screen.blit(bg, (0, 0))

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

```python
# 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 -= 2
    # 判断飞机的位置
    if hero_rect.y + hero_rect.height <= 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()
```

---

016. 事件监听-01-基本概念和event模块的get方法

事件 `event`

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

监听

- 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

- `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

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

```
# 游戏循环
while True:

    # 设置屏幕刷新帧率
    clock.tick(60)

    # 事件监听
    for event in pygame.event.get():

        # 判断用户是否点击了关闭按钮
        if event.type == pygame.QUIT:
            print("退出游戏...")

            pygame.quit()

            # 直接退出系统
            exit()
```

---

017. 事件监听-02-监听事件并且退出游戏

```
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
bg = pygame.image.load('./images/background.png')
screen.blit(bg, (0, 0))

# 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
```

```python
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)
    # 监听事件
    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 卸载所有模块
            pygame.quit()
            # exit() 直接终止当前正在执行的程序
            exit()
    # 2. 修改飞机的位置
    hero_rect.y -= 2
    # 判断飞机的位置
    if hero_rect.y + hero_rect.height <= 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()
```

---

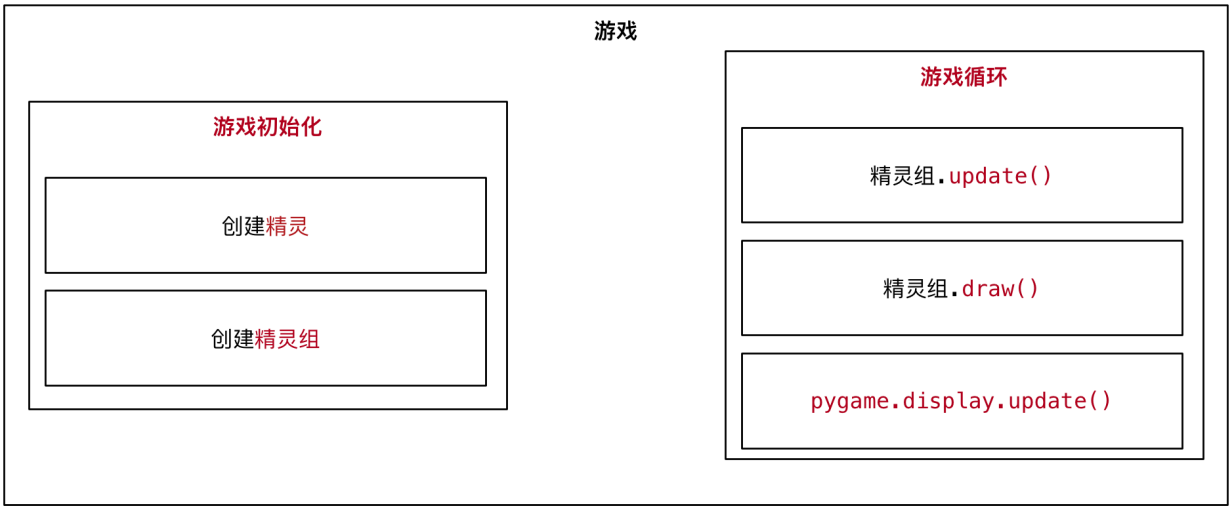
018. 精灵和精灵组-01-基本概念

精灵 和 **精灵组**

- 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
- 为了简化开发步骤，`pygame` 提供了两个类
    - `pygame.sprite.Sprite` —— 存储 **图像数据 image** 和 **位置 rect** 的 **对象**
    - `pygame.sprite.Group`

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



精灵

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

精灵组

- 一个 **精灵组** 可以包含多个 **精灵** 对象
- 调用 **精灵组** 对象的 `update()` 方法
  - 可以 **自动** 调用 **组内每一个精灵** 的 `update()` 方法
- 调用 **精灵组** 对象的 `draw(屏幕对象)` 方法
  - 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

019. 精灵和精灵组-02-自定义精灵子类需求分析

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

注意

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

属性

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

方法

- `update` 每次更新屏幕时在游戏循环内调用
  - 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

020. 精灵和精灵组-03-派生精灵子类代码实现

```
import pygame
```

```python
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
```

注意：如果类不是继承自 `object`，要主动调用父类的初始化方法

---

021. 精灵和精灵组-04-创建敌机并且实现敌机动画

需求

- 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

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

职责

- 精灵
    - 封装 **图像 image**、**位置 rect** 和 **速度 speed**
    - 提供 `update()` 方法，根据游戏需求，**更新位置 rect**
- 精灵组
    - 包含 **多个 精灵对象**
    - `update` 方法，让精灵组中的所有精灵调用 `update` 方法更新位置

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

实现步骤

- 1) 导入 `plane_sprites` 模块

```
from plane_sprites import *
```

- 2) 修改初始化部分代码

```
# 创建敌机精灵和精灵组
enemy1 = GameSprite("./images/enemy1.png")
enemy2 = GameSprite("./images/enemy1.png", 2)
enemy2.rect.x = 200
enemy_group = pygame.sprite.Group(enemy1, enemy2)
```

- 3) 修改游戏循环部分代码

```
# 让敌机组调用 update 和 draw 方法
enemy_group.update()
enemy_group.draw(screen)

# 更新屏幕显示
pygame.display.update()
```

最终代码

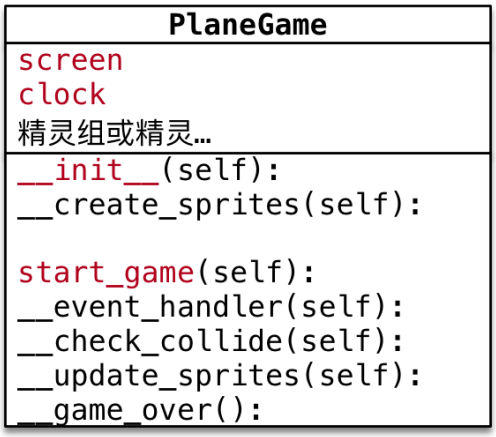
```
import pygame
from plane_sprites import *

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))

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

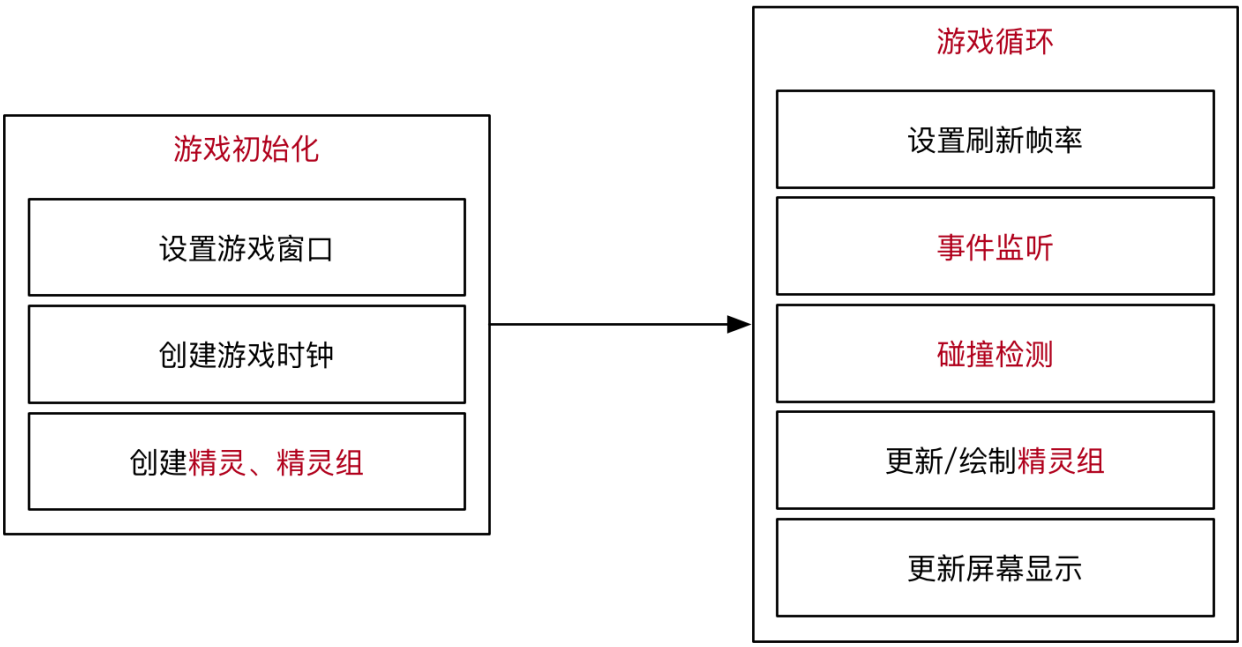
```python
# 可以在所有绘制工作完成之后，统一调用update方法
pygame.display.update()

# 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)

# 创建敌机的精灵
enemy = GameSprite('./images/enemy1.png')
enemy1 = GameSprite('./images/enemy1.png', 2)
# 创建敌机的精灵组
enemy_group = pygame.sprite.Group(enemy, enemy1)

# 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)
    # 监听事件
    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 卸载所有模块
            pygame.quit()
            # exit() 直接终止当前正在执行的程序
            exit()
    # 2. 修改飞机的位置
    hero_rect.y -= 2
    # 判断飞机的位置
    if hero_rect.y + hero_rect.height <= 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)
    # 让精灵组调用两个方法
    # update - 让组中的所有精灵更新位置
    enemy_group.update()
    # draw - 在screen上绘制所有的精灵
    enemy_group.draw(screen)
    # 4. 调用update方法更新显示
    pygame.display.update()

pygame.quit()
```

022. 框架搭建-01-明确飞机游戏类的设计

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：





提示

- 根据 **职责** 封装私有方法，可以避免某一个方法的代码写得太过冗长
- 如果某一个方法编写的太长，既不好阅读，也不好维护！
- **游戏初始化** —— `__init__()` 会调用以下方法：

方法	职责
`__create_sprites(self)`	创建所有精灵和精灵组

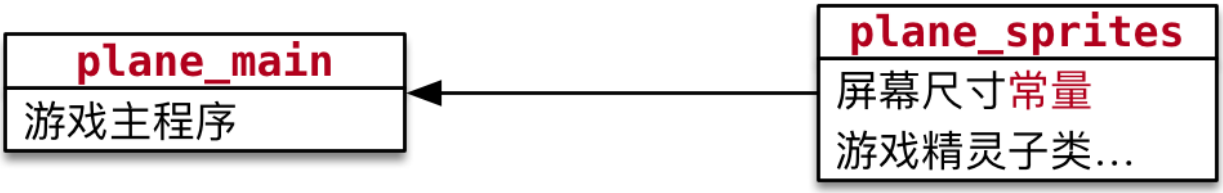
- **游戏循环** —— `start_game()` 会调用以下方法：

方法	职责
`__event_handler(self)`	事件监听
`__check_collide(self)`	碰撞检测 —— 子弹销毁敌机、敌机撞毁英雄
`__update_sprites(self)`	精灵组更新和绘制
`__game_over()`	游戏结束

---

023. 框架搭建-02-明确文件职责准备主游戏类

明确文件职责



- `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()
```

> 【!】**注意**：
> `PyCharm` 小技巧
> 输入 `main` 然后 `Enter` 或者 `Tab`
> 会自动补全

```
if __name__ == '__main__':
```

---

024. 框架搭建-03-游戏初始化

- 完成 `__init__()` 代码如下

```
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')
        # 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

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

```python
if __name__ == '__main__':
    # 创建游戏对象
    game = PlaneGame()
    # 启动游戏
    game.start_game()
```

---

025. 框架搭建-04-使用常量定义游戏窗口大小

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**

> 【?】魔法数字是什么？ magic number?

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)
```

最终代码

```
import pygame
from plane_sprites import *

# 屏幕大小的常量
SCREEN_RECT = pygame.Rect(0, 0, 480, 700)


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

    def __init__(self):
        print('游戏初始化')
        # 1. 创建游戏的窗口
        self.screen = pygame.display.set_mode(SCREEN_RECT.size)
        # 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

    def start_game(self):
        print('游戏开始...')
        while True:
            pass


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

026. 框架搭建-05-搭建启动游戏方法结构

`D:\code\python-fullstack\darkhorse\飞机大战\plane_sprites.py`

```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
```

`D:\code\python-fullstack\darkhorse\飞机大战\plane_main.py`

```python
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')
        # 1. 创建游戏的窗口
        self.screen = pygame.display.set_mode(SCREEN_RECT.size)
        # 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

    def start_game(self):
        print('游戏开始...')
        while True:
            # 1. 设置刷新帧率
            self.clock.tick(FRAME_PER_SEC)
            # 2. 事件监听
            self.__event_handler()
            # 3. 碰撞检测
            self.__check_collide()
            # 4. 更新/绘制精灵组
            self.__update_sprites()
            # 5. 更新显示
            pygame.display.update()

    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()

    def __check_collide(self):
        pass

    def __update_sprites(self):
        pass

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


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

完成于`201810101541`