Pygame 安装

使用管理员权限打开“命令提示符”，在“命令提示符”输入命令：

pip install pygame -i https://mirrors.aliyun.com/pypi/simple/

一、Pygame 入门

2.1 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。

`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()
```

二、游戏中的坐标系

2.1 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): 起始点的横纵坐标

(width, height): 矩形的宽度和高度

2.2 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
```

下面我们通过实际代码使用 `Rect` 类

```
import pygame

rect = pygame.Rect(100, 500, 120, 125)

print("英雄的原点 ({}, {})".format(rect.x, rect.y))
print("英雄的尺寸 ({}, {})".format(rect.width)
print(rect.size)
```

三、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**

`pygame.display.set_mode()`：初始化游戏显示窗口

`pygame.display.update()`：刷新屏幕内容显示

set_mode()方法

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

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：

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

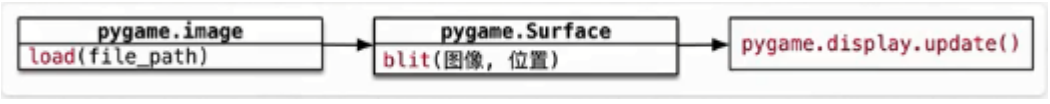
```
import pygame

pygame.init()
# 创建游戏的窗口 480 * 700
screen = pygame.display.set_mode((480, 700))
while True:
    pass
pygame.quit()
```

四、图像的绘制

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：

- 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



4.1 绘制背景图片

```
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()
```

4.2 绘制飞机图像

```
# 绘制飞机
hero = pygame.image.load("./images/me1.png")
screen.blit(hero, (150, 300))
pygame.display.update()
```

4.3 update()方法

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。

所以我们可以在 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。

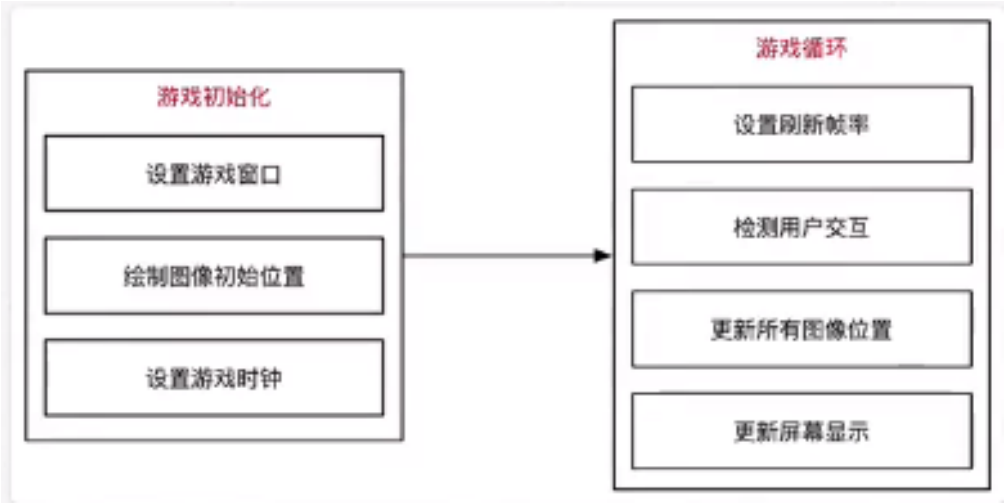
五、游戏循环和游戏时钟

5.1 游戏动画的的实现原理

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

5.2 游戏循环



5.3 游戏时钟

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

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

# 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(1) #1s 中一次
    pass
```

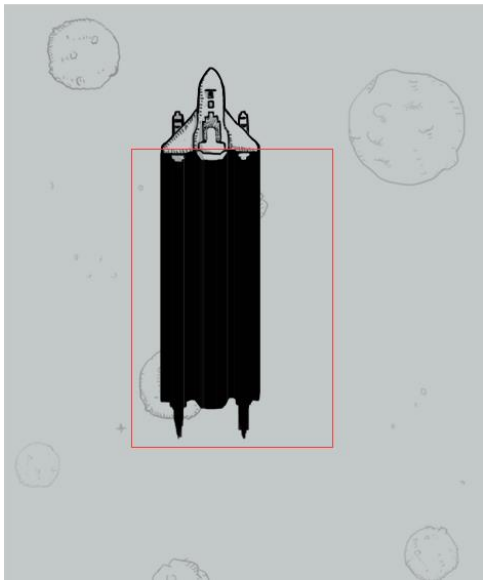
5.4 简单动画的实现

我们之前了解到，游戏中的可见元素都是以矩形来表示的，所以飞机也可以用一个矩形来表示。

hero_rect = pygame.Rect(150, 300, 102, 126)
那么如何实现飞机的移动呢？将飞机这个矩形显示的横纵坐标进行修改即可，例如：
hero_rect.y -= 1 则表示矩形向上移动一个坐标（思考：为什么是向上呢？）

```
# 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(hero, hero_rect)

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

运行程序后我们发现执行结果是这样的：



为什么呢？因为显示新的飞机的时候屏幕中还有上次显示的飞机图片。
如何解决呢？在显示新的飞机之前，重新显示背景图片将上一次显示的飞机覆盖

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

5.5 练习

编写代码实现，当飞机飞到屏幕的顶部的时候，从屏幕的最下方重新进入。

六、游戏中的监听事件

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

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

```
# 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

    # 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()
```

```
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()
```

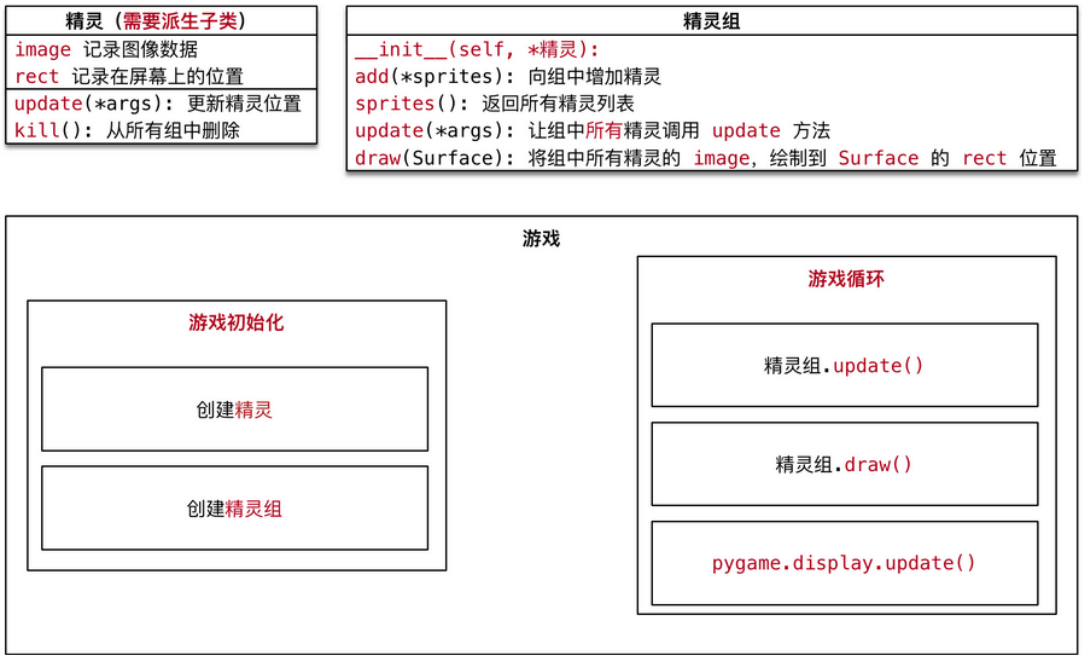
七、精灵和精灵组

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



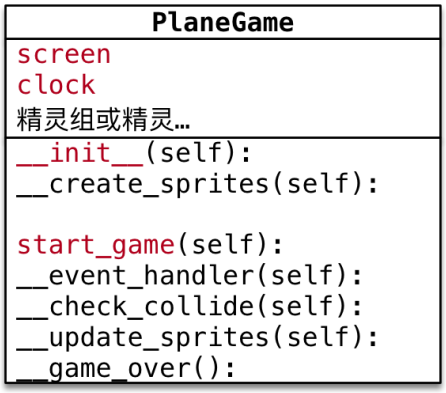
7.1 设计一个类来描述敌机

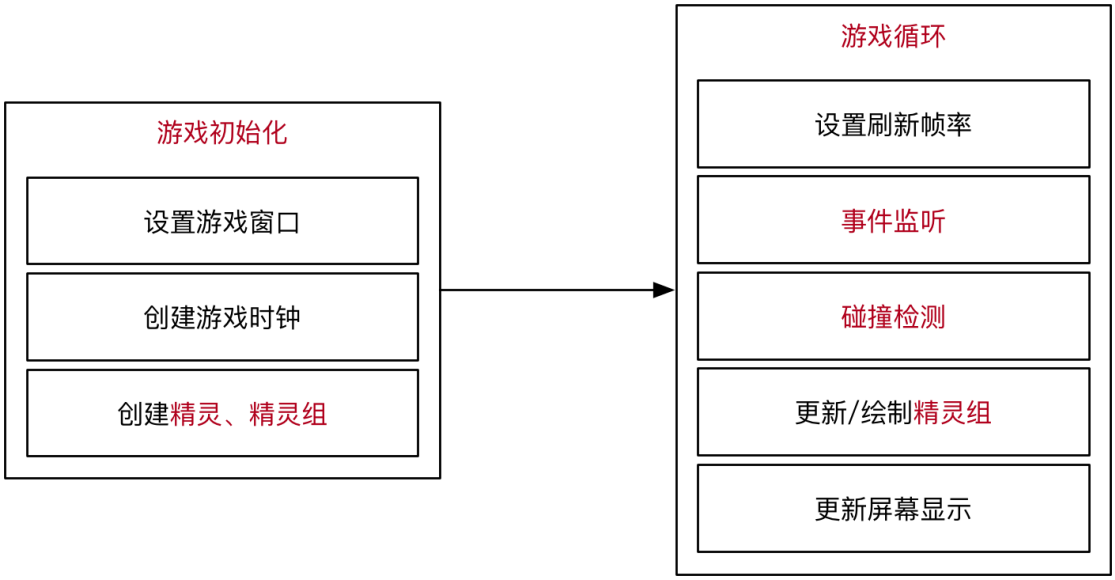
```
class GameSprite(pygame.sprite.Sprite):
    """飞机大战游戏精灵"""
    def __init__(self, image_name, speed=1):
        # 调用父类的初始化方法
        super().__init__()
        # 定义对象的属性
        self.image = pygame.image.load(image_name)
        self.rect = self.image.get_rect()
        self.speed = speed
    def update(self):
        # 在屏幕的垂直方向上移动
        self.rect.y += self.speed
```

7.2 敌机与英雄飞机动画效果实现

```
import pygame
from plane_sprites import *
# 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)

    # 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)
  # 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()
```

八、封装一个游戏类






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

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

    def start_game(self):
        print("游戏开始...")


if __name__ == '__main__':

    # 创建游戏对象
    game = PlaneGame()

    # 启动游戏
    game.start_game()
```

下面我们对游戏进行进一步的初始化：

```
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...")


if __name__ == '__main__':

    # 创建游戏对象
    game = PlaneGame()

    # 启动游戏
    game.start_game()
```

下面我们来实现 start_game 中的逻辑：

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

        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/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():
            Pass

    def __check_collide(self):
        pass

    def __update_sprites(self):
        Pass
```

九、创建背景精灵类

思考：如何实现飞机向上移动的效果？

飞机其实是不动的，将背景图片向下移动就可以实现

```
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().__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，将图像设置到屏幕的上方
        if self.rect.y >= 700:
            self.rect.y = -self.rect.height
```

背景精灵创建完成以后，下面我们将背景动画效果进行实现：

```
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):
        #  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)

    def start_game(self):
        print("游戏开始...")

        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/绘制精灵组
            self.__update_sprites()
            # 5. 更新显示
            pygame.display.update()

    def __event_handler(self):
        pass

    def __check_collide(self):
        pass

    def __update_sprites(self):
        self.back_group.update()
        self.back_group.draw(self.screen)

if __name__ == '__main__':

    # 创建游戏对象
    game = PlaneGame()

    # 启动游戏
    game.start_game()
```

我们来思考一下以下代码是否有不合理的地方？

```
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)
```

接下来，我们优化一下我们的背景精灵类：

```
class Background(GameSprite):
    """游戏背景精灵"""
    def __init__(self, is_alt=False):
        # 1. 调用父类方法实现精灵的创建(image/rect/speed)
        super().__init__("./images/background.png")
        # 2. 判断是否是交替图像，如果是，需要设置初始位置
        if is_alt:
```

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

    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 >= 700:
            self.rect.y = -self.rect.height
```

修改__create_sprites 方法：

```
def __create_sprites(self):
        #  创建背景精灵和精灵组
        bg1 = Background()
        bg2 = Background(True)
        self.back_group = pygame.sprite.Group(bg1, bg2)
```

通过对背景精灵类的优化，我们就将和背景图片操作相关的代码全部整合到了一个类中了。

十、敌机出场的实现

首先我们运行 **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

下面我们学习如何使用定时器每隔一段时间执行某些操作。

10.1 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**

```
python 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

10.2 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：

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

代码：“08_创建敌机定时器事件”

十一、设计一个敌机类 Enemy

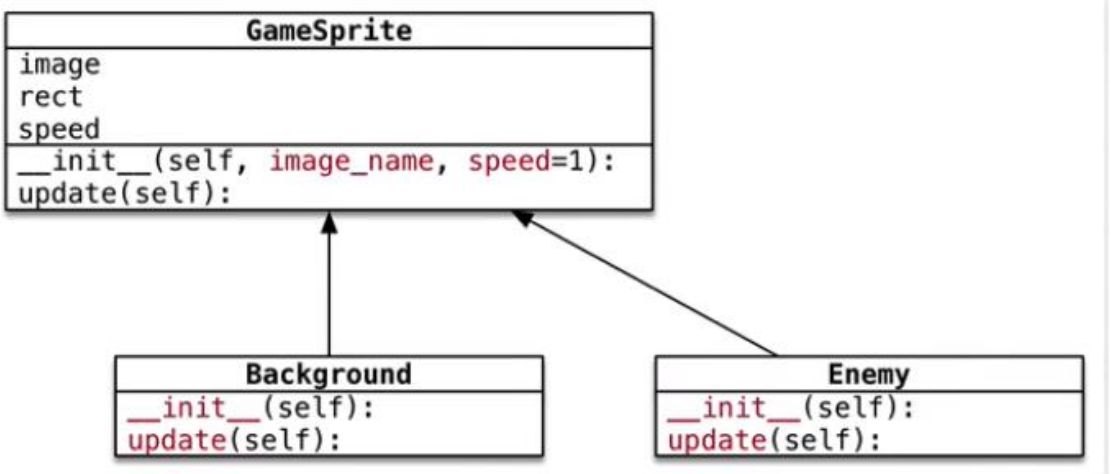
首先我们来回顾一下敌机的出现规律：

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



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

下面我们来设计一个敌机类：

- 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，同时指定敌机图片
        super().__init__("./images/enemy1.png")

        # 2. 指定敌机的初始随机速度

        # 3. 指定敌机的初始随机位置

    def update(self):

        # 1. 调用父类方法，保持垂直方向的飞行
        super().update()

        # 2. 判断是否飞出屏幕，如果是，需要从精灵组删除敌机
        if self.rect.y >= SCREEN_RECT.height:
            print("飞出屏幕，需要从精灵组删除...")
```

代码：”09_准备敌机类”
接下来，我们需要创建敌机：

创建敌机

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

接下来我们修改 `plane_main` 的 `__create_sprites` 方法

```
def __create_sprites(self):

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

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

        # 创建敌机的精灵组
        self.enemy_group = pygame.sprite.Group()
```

因为敌机是通过定时器进行创建的，所以 __create_sprites(self)方法中是不需要创建敌机的，我们值需要在定时器事件中创建敌机即可，所以我们还需要修改__event_handler 方法：

```
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...")
                # 创建敌机精灵
                enemy = Enemy()

                # 将敌机精灵添加到敌机精灵组
                self.enemy_group.add(enemy)
```

精灵和精灵组在创建完成以后，如果需要显示在游戏界面中我们该怎么做呢？我们还需要修改：__update_sprites 方法

```
def __update_sprites(self):

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

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

代码：“10_创建并显示敌机”

在上一个代码中我们发现敌机出现的位置和速度都是一样的，接下来我们需要对敌机的速度和出现的位置进行随机处理：

首先我们先对敌机出现的速度进行随机处理，修改 plane_sprites.py 中 Enemy 类的 init 方法：

```
def __init__(self):

    # 1. 调用父类方法，创建敌机精灵，同时指定敌机图片
    super().__init__("./images/enemy1.png")

    # 2. 指定敌机的初始随机速度 1 ~ 3
    self.speed = random.randint(1, 3)
```

最后我们对敌机出现的位置进行随机处理：继续修改 init 方法：

```
def __init__(self):
        # 1. 调用父类方法，创建敌机精灵，同时指定敌机图片
        super().__init__("./images/enemy1.png")

        # 2. 指定敌机的初始随机速度 1 ~ 3
        self.speed = random.randint(1, 3)

        # 3. 指定敌机的初始随机位置
        self.rect.bottom = 0

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

代码：“11_敌机的随机速度和位置”

当敌机飞出屏幕后我们需要销毁敌机对象？为甚呢？

`__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**

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

修改 update 方法：

```
def update(self):

        # 1. 调用父类方法，保持垂直方向的飞行
        super().update()
```

```
            # 2. 判断是否飞出屏幕，如果是，需要从精灵组删除敌机
            if self.rect.y >= SCREEN_RECT.height:
                print("飞出屏幕，需要从精灵组删除...")
                # kill 方法可以将精灵从所有精灵组中移出，精灵就会被自动销毁
                self.kill()

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

代码：“12_飞出屏幕销毁敌机”

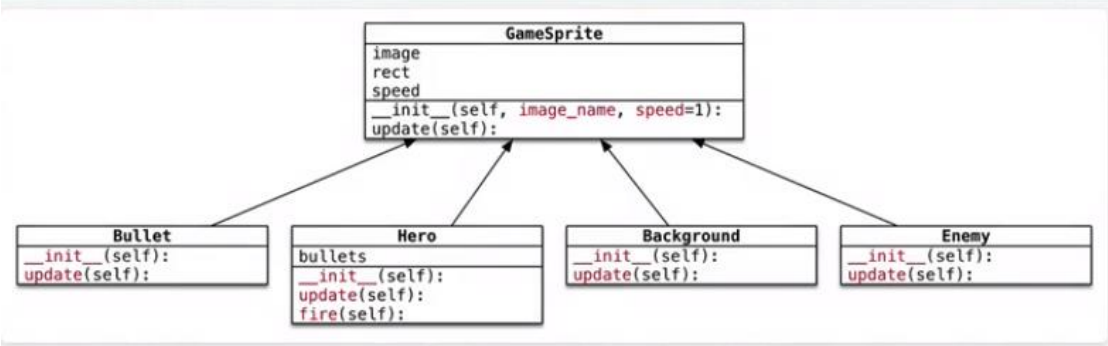
十二、设计英雄飞机和子弹类

英雄需求

- 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**
- **英雄** 每隔 `0.5` 秒发射一次子弹，每次 **连发三枚子弹**
- **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

- 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



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

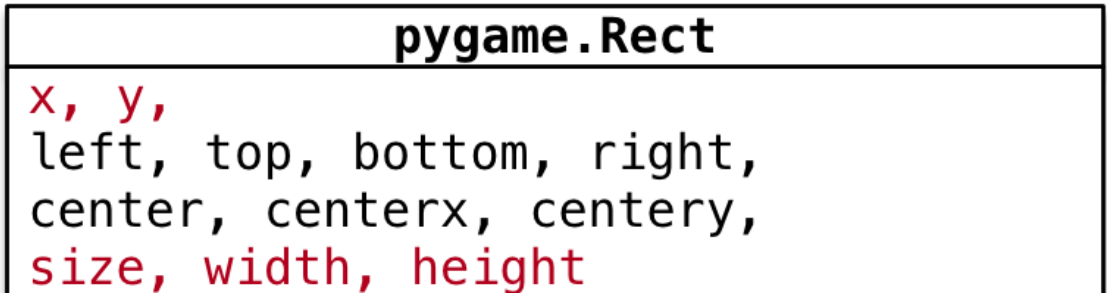
12.1 英雄类 Hero 的实现

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**



思考：如何设置英雄飞机的初始位置呢？

centerx 是矩形的中心位置

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

    def __init__(self):

        # 1. 调用父类方法，设置 image&speed
        super().__init__("./images/me1.png", 0)

        # 2. 设置英雄的初始位置
```

```
        self.rect.centerx = SCREEN_RECT.centerx
        self.rect.bottom = SCREEN_RECT.bottom - 120
```

Hero 类设计完成以后，下面我们需要将英雄飞机显示在屏幕上
第一步，我们需要构造英雄飞机
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**

我们可以通过修改 PlaneGame 类中的__create_sprites 创建英雄的精灵和精灵组

```
def __create_sprites(self):

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

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

        # 创建敌机的精灵组
        self.enemy_group = pygame.sprite.Group()

        # 创建英雄的精灵和精灵组
        self.hero = Hero()
        self.hero_group = pygame.sprite.Group(self.hero)
```

第二步：在 `__update_sprites`，让 **英雄精灵组** 调用 `update` 和 `draw` 方法

```
def __update_sprites(self):

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

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

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

代码：“14_绘制英雄”

12.2 捕获按键控制飞机的左右移动

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`

```
        if keyspressed[pygame.KRIGHT]:  #判断向右的键是否被按下
            Pass
```

我们可以修改__event_handler 函数:

```
keyspressed = pygame.key.get_pressed()
if keyspressed[pygame.K_RIGHT]:
    print("向右移动")
    self.hero.speed = 2
elif keyspressed[pygame.K_LEFT]:
    print("向左移动")
    self.hero.speed = -2
else:
    self.hero.speed = 0
```

那如何才能移动英雄飞机呢？
回顾一下 GameSprite 类中的 update 方法的作用是什么？更新精灵对象的位置！
所以我们可以在 Hero 类中重写 update 方法：

```
def update(self):
    self.rect.x += self.speed
    # 控制英雄不能离开屏幕
    if self.rect.x < 0:
        self.rect.x = 0
    elif self.rect.right > SCREEN_RECT.right:
        self.rect.right = SCREEN_RECT.right
```

十三、发射子弹设计

需求：
1）**英雄** 每隔 `0.5` 秒发射一次子弹，每次 **连发三枚子弹**
2）**子弹** 从 **英雄** 的正上方发射 **沿直线** 向 **上方** 飞行
3）飞出屏幕后，需要从 **精灵组** 中删除

13.1 实现 0.5s 发射一次子弹

如何实现每隔 `0.5` 秒发射一次子弹，每次 **连发三枚子弹？**
通过定时器来实现：
1）定义一个子弹发射事件：

```
HERO_FIRE_BULLET = pygame.USEREVENT+1
```

2）在 PlaneGame 类的__init__方法中设置一个定时器

```
pygame.time.set_timer(HERO_FIRE_BULLET, 500)
```

3）__event_handler 方法中判断 HERO_FIRE_BULLET 事件是否发生，如果发生则发射子弹

```
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...")

        # 创建敌机精灵
        enemy = Enemy()
        # 将敌机精灵添加到敌机精灵组
        self.enemy_group.add(enemy)
    elif event.type == HERO_FIRE_BULLET:
        self.hero.fire()
```

4）在 Hero 类中封装一个方法 fire 来实现子弹的发射

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

13.2 子弹类的实现

同样的子弹类 Bullet 也可以继承自 GameSprite 类:

```
class Bullet(GameSprite):
    """子弹精灵"""
    def __init__(self):
        # 调用父类方法，设置子弹图片，设置初始速度
        super().__init__("./images/bullet1.png", -2)

    def update(self):
        # 调用父类方法，让子弹沿垂直方向飞行
        super().update()

        # 判断子弹是否飞出屏幕
        if self.rect.bottom < 0:
            self.kill()

    def __del__(self):
        print("子弹被销毁...")
```

子弹类设计完成以后，下面我们需要在 PlaneGame 类的 fire 方法中创建子弹实现子弹的发射：
1）在 Hero 的__init__方法中创建子弹精灵组

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

2）在 Hero 类的 fire 方法中创建 3 颗子弹

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

    for i in range(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)
```

14、碰撞检测

14.1 子弹摧毁敌机

两个精灵组 中 **所有的精灵** 的碰撞检测

```
python groupcollide(子弹, 敌机, True, False, 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**

14.2 敌机摧毁英雄飞机

pygame.sprite.spritecollide()

判断 **某个精灵** 和 **指定精灵组** 中的精灵的碰撞

```
python 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**

```
# 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()
```