飞机大战

目的

- 综合 **Python初级课** 知识
- 体验 **游戏** 实战开发

准备

pygame 游戏模块

官网地址: https://www.pygame.org/

> pygame 是专门为做游戏而准备的模块, 可以直接下载并开发游戏

安装pygame

- python安装包

- pip安装包
- pygame安装包

Python安装包

下载地址:

windows版: https://www.python.org/downloads/windows/

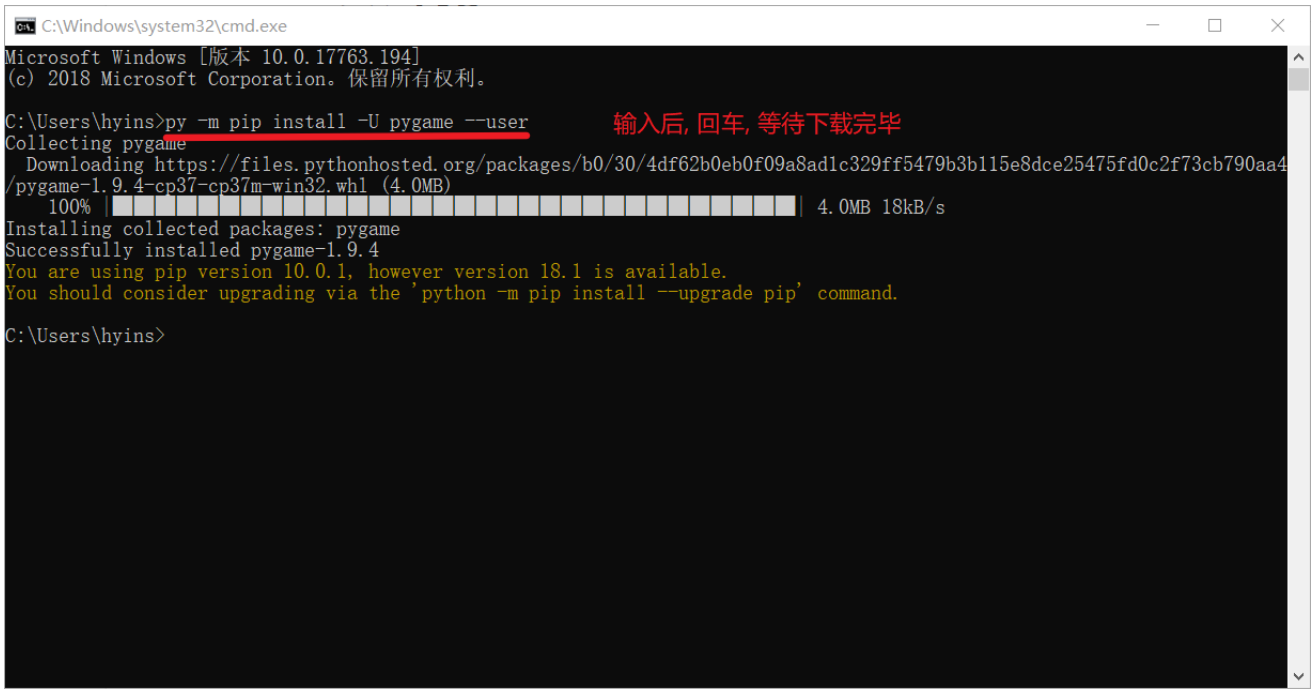
pip 安装包

Python 只要安装的 Python3.4 或 Python3.4 以后, 都自带pip

pygame安装包

1. 在线安装

- 按住 Win + R (win就是键盘左下角4个格子的键, 也叫开始菜单键)
- 输入 cmd
- 输入 `py -m pip install -U pygame --user`



---

2. 离线安装

2.1 安装whell

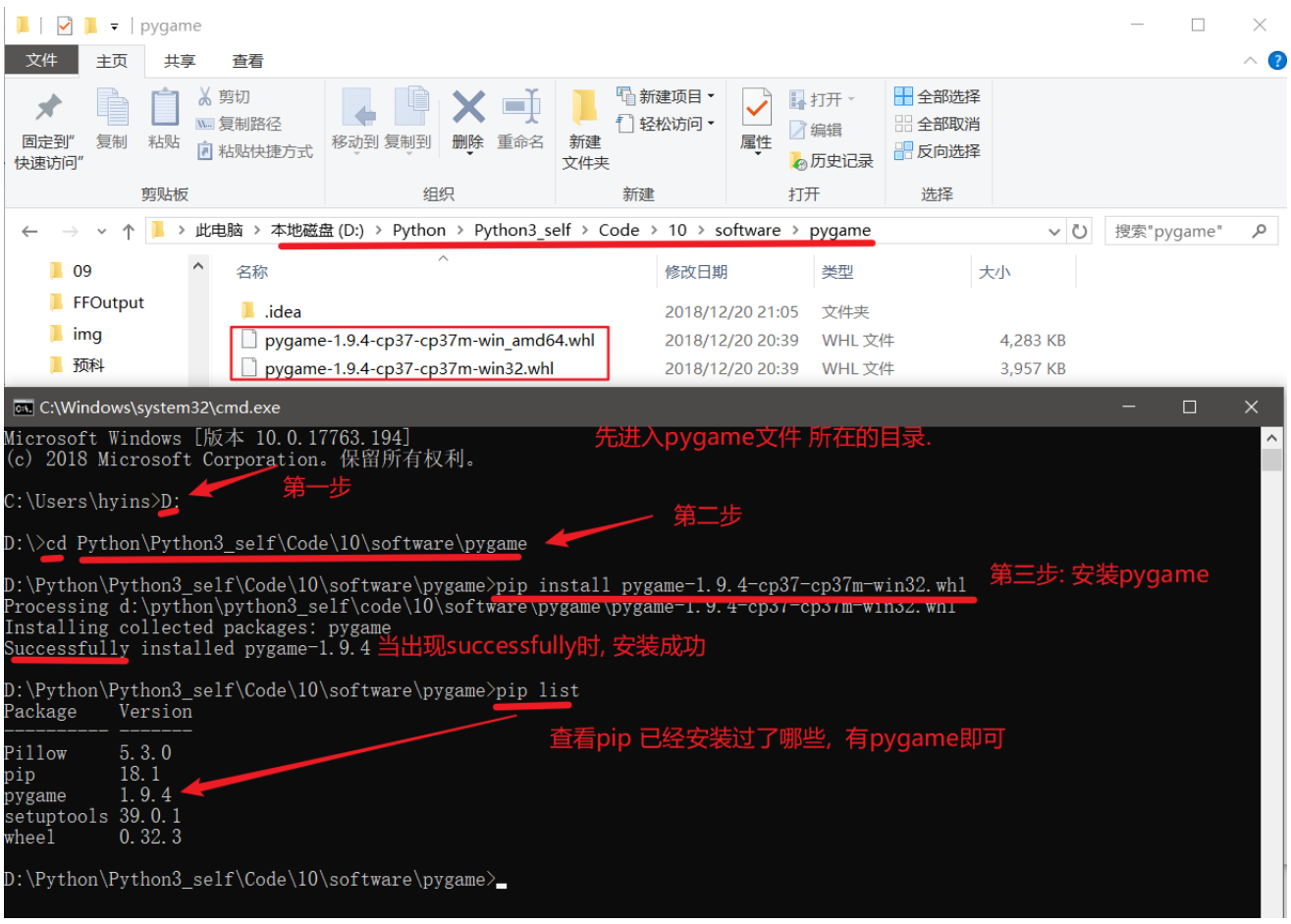
- 进入cmd
- 输入 pip install wheel
- 输入 pip list 查看是否有wheel, 有则证明安装成功

2.2 下载pygame

- 下载地址: https://www.lfd.uci.edu/~gohlke/pythonlibs/

  建议 32位 和 64位 都下载一个

- **安装pygame**

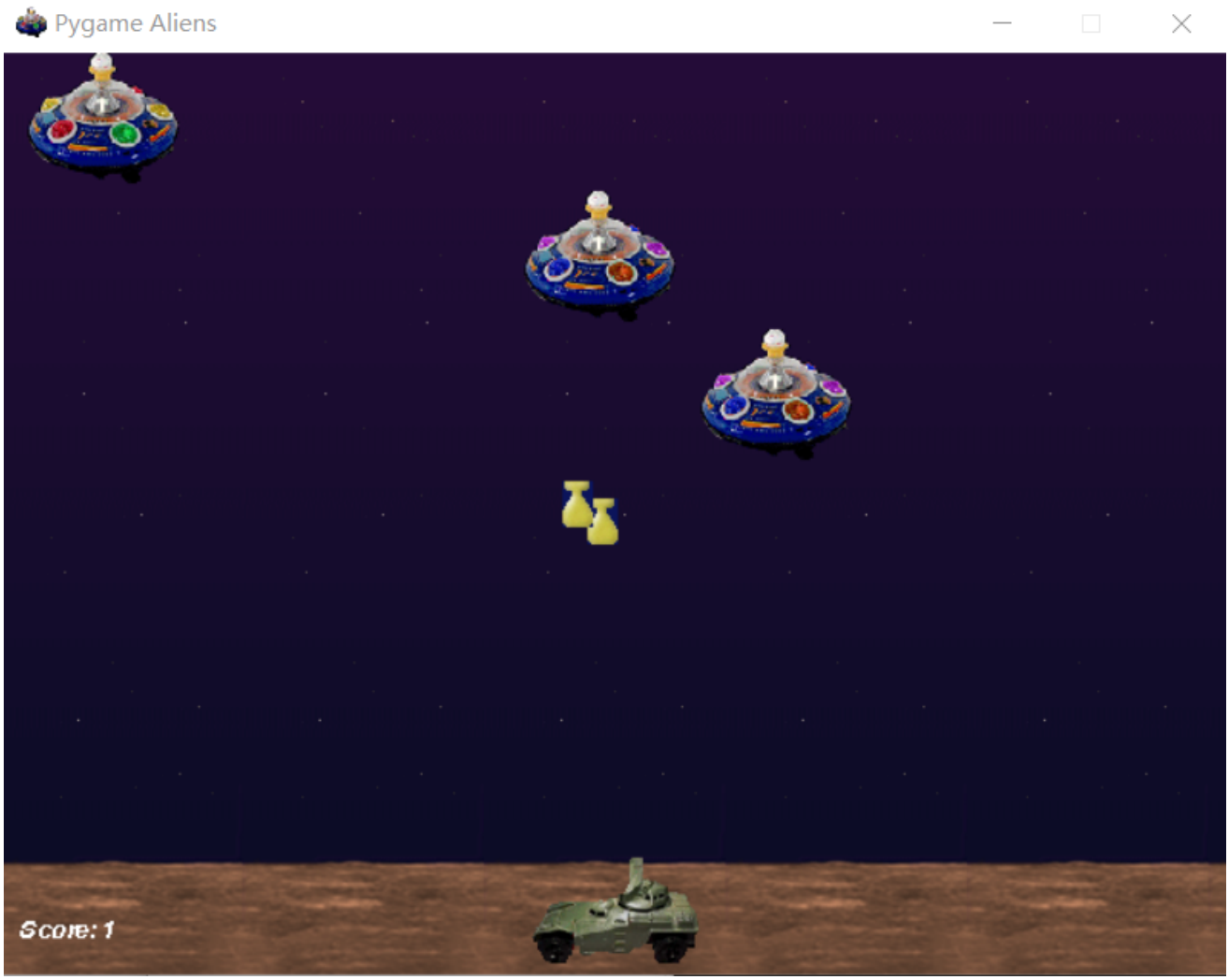


- **检测是否安装成功**

  在cmd 中输入:

```
py -m pygame.examples.aliens
```

如果弹出一个游戏界面, 则证明已成功安装



设计游戏

飞机游戏基本了解

- 设计一个 带有**游戏背景**的 **游戏窗口**
- 在 游戏窗口中, 放置**我方飞机** 和 **敌方飞机**的 **图片**
- 根据 **用户的操作** 移动我方飞机**图片**, 产生动画效果
- 根据 **图片之间** 是否发生**重叠**, 判断敌方飞机是否被**摧毁** 等情况

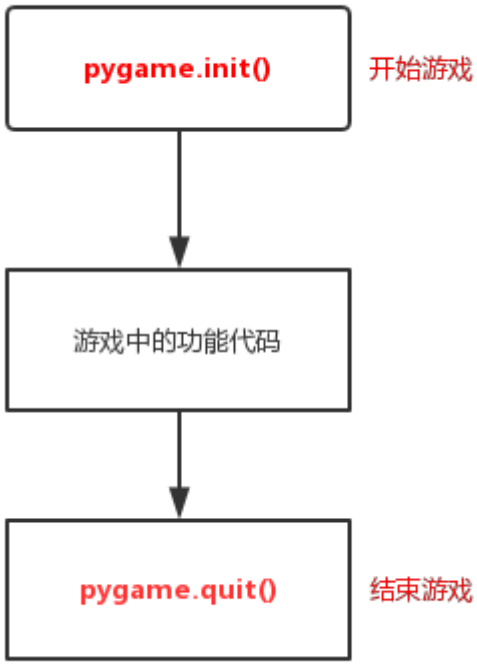
初始化游戏

首先, 要明白, 在创建游戏窗口之前, 需要先**准备好**游戏需要用到的**模块**

步骤:

- 初始化游戏模块
- 退出游戏, 并卸载模块

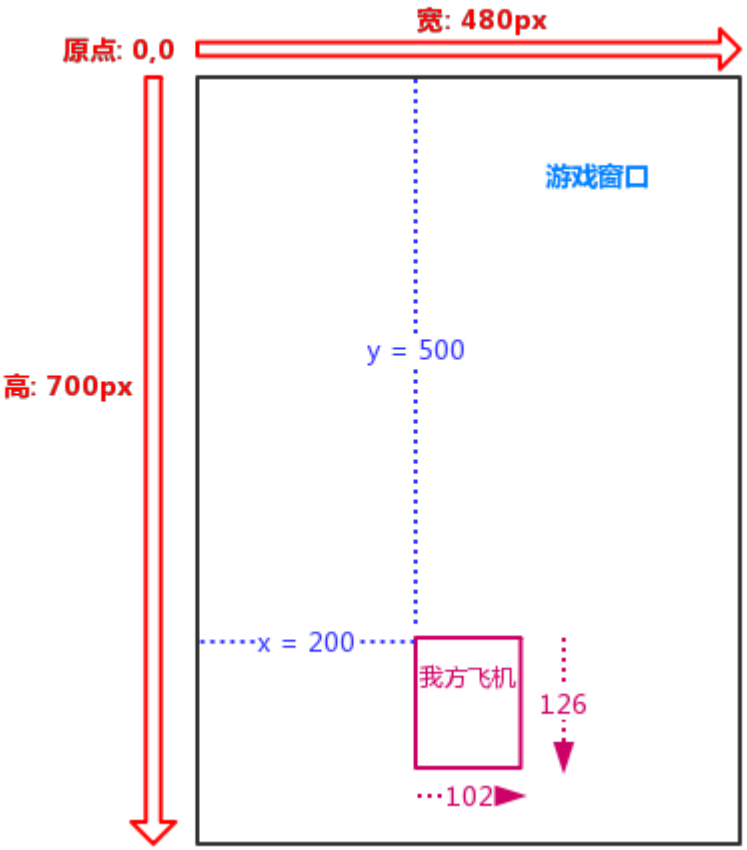
方法	功能
pygame.init()	加载并初始化pygame所有模块, 准备开始游戏
pygame.quit()	卸载pygame所有模块, 准备结束游戏



创建游戏窗口

创建窗口之前, 先了解坐标系

- 左上角, 作为**原点**, **坐标** `0,0`
- 原点**向右**, 逐渐增加, 是为 **x轴**
- 原点**向下**, 逐渐增加, 是为 **y轴**



通过上图, 可以看出, 我们准备设计一个 **宽: 480 高:700** 的窗口

在pygame中, 通过 **pygame.display** 来**创建** 和 **管理窗口**

方法	功能	参数
pygame.display.set_mode()	初始化窗口	resolution, flags, depth
pygame.display.update()	刷新窗口内容	

pygame.display.set_mode(resolution=[0,0], flags=0, depth=0)

- 参数
  - resolution 指定窗口的**分辨率**, 即 **宽** 和 **高**, 默认与当前屏幕一致
  - flags 指定窗口的附加选项.
    - pygame.FULLSCREEN 控制全屏
    - pygame.HWSURFACE 控制是否进行硬件加速 (全屏下才能使用)
    - pygame.RESIZABLE 控制窗口是否可以调节大小
    - pygame.NOFRAME 控制窗口无边框, 无按钮

```
* 如需多个一起使用，则以 `|`  竖线隔开
* 若无特殊要求，建议使用默认的0
```

- depth 指定颜色深度
  - 默认 Pygame 会根据当前操作系统选择最好和最快的颜色深度 (推荐)
- 返回值
  - 返回 **Surface** 对象. 可以理解为 屏幕 或 画布

```
案例:
    screen = pygame.display.set_mode((480,700))
    创建一个 480*700 的窗口
```

防闪退

上一步创建完 窗口后, 就立马闪退了.

原因: 创建完窗口之后, 没有太多其他代码运行了, **程序将要结束**了, 所以**窗口跟着一起结束**了.

解决: **不让程序结束**, 窗口不就不会消失了嘛

```
案例:
    while True:
        pass
    游戏无限循环while代码
    这样程序永远不会结束，窗口也就永远存在了
```

关闭窗口

上一步完成后, 窗口是不会闪退, 但也没其他操作, 点关闭窗口也没有反应

原因: 程序一直在 **无限循环** 中呢, 根本**来不及响应**你的关闭操作

解决: 监听用户的关闭窗口操作, 当用户点击关闭时, 就直接**退出游戏, 卸载模块**

通过**pygame.event** 可以**监听**用户的所有操作事件

监听用户是否点击 "关闭" 按钮

```
for event in pygame.event.get():          获取所有的事件
    if event.type == pygame.QUIT:         判断是否 关闭按钮 事件
        print('退出游戏 ...')             提示退出
        pygame.quit()                     卸载模块
        exit()                            退出程序
```

要点分析

- pygame.event.get() 可以获取到所有的操作. 并返回 **事件列表**
- 其中 **列表值.type** 能获取当前具体事件

event方法

方法	作用	参数
pygame.event.get()	获取所有事件	无

pygame.event.get()

- 返回值
  - 列表 [<Event (5-MouseButtonDown {'pos': (51, 683), 'button': 1})>]
    - **type**: MouseButtonDown 按下鼠标
    - **pos**: (51, 683) 鼠标坐标 x: 51 y: 683
    - **button**: 1 左击: 1 滚轮:2 右击:3
    - 其余的有待挖掘 ...

type事件类型	作用	参数
Quit	用户按下关闭按钮	无
KeyDown	按下键盘	unicode,key,mod
KeyUp	松开键盘	key, mod
MouseMotion	鼠标移动	pos，rel，buttons
MouseButtonDown	按下鼠标	pos，button
MouseButtonUp	松开鼠标	pos，button
ActiveEvent	pygame被激活或者隐藏	gain，state
USEREVENT	用户自定义事件, 生成事件id	

以上 type事件 真正应用时, 要**全部大写**

pygame 属性

属性	作用
pygame.QUIT	关闭按钮
pygame.KEYDOWN	键盘按下
pygame.KEYUP	键盘松开
pygame.K_LEFT	方向键: 左
pygame.K_RIGHT	方向键: 右
pygame.K_UP	方向键: 上
pygame.K_DOWN	方向键: 下
pygame.MouseMotion	鼠标移动

绘制窗口背景

在大家平时玩的游戏中, 大多数的**元素** 都是**图像**, 例如: 人, 小怪, 飞机, NPC ...

想要在 **窗口**看到 **图像**的**三步骤**

- 通过 `pygame.image.load()` 来加载图像
- 通过 `Surface对象.blit()` 设置图像在窗口中的坐标
- 通过 `pygame.display.update()` 刷新窗口屏幕, 来显示最新的窗口内容

案例

```
    1. 加载背景图片
    background = pygame.image.load('image/background.png')

    2. 设置背景图片的 坐标
    screen.blit(background, (0,0))

    3. 刷新窗口屏幕
    pygame.display.update()
```

> **步骤 1:**

pygame.image.load(imgPath)

- 功能
  - 加载图像
- 参数
  - imgPath: 图像地址

- 返回值
  - Surface 对象

> **步骤 2:**

Surface对象1.blit(Surface对象2, pos)

- 功能
  - 设置坐标
- 参数
  - Surface: 准备在**大Surface对象1**中绘制 **小Surface对象2**
  - pos: 以容器形式 设置坐标.
    - (5,10) 以元组形式设置坐标 x:5 y:10
    - [5,10] 以列表形式设置坐标 x:5 y:10
- 返回值
  - Rect矩形对象

> **步骤 3:**

pygame.display.update()

- 功能:
  - 刷新屏幕, 显示最新的界面
- 参数:
  - 无
- 返回值:
  - 无

要点分析

- 第二步: 设置背景图片的 坐标
  - screen: 是保存前面创建窗口的变量
  - background: 是刚刚加载的图像
  - 将background背景 绘制在 screen窗口中, 从screen窗口的原点(0,0) 开始绘制.
- 第三步: 刷新窗口
  - 其实, 任何**改变画面**的操作后, 都需要刷新屏幕. 因为游戏就是**帧游戏**.

绘制 我方英雄飞机

这里给大家提供两种方案:

- 与绘制窗口背景方式一致

- 抠图方式

方案1:

```
1. 加载飞机图片
plane = pygame.image.load('image/single.png')

2. 设置背景图片的 坐标
   将飞机绘制在窗口中，从窗口200,500点开始绘制
screen.blit(plane, (200,500))

3. 刷新窗口屏幕
pygame.display.update()
```

方案2:

```
1.1 加载一张大图（含有各种飞机）
plane_img = pygame.image.load('image/shoot.png')

1.2 设置矩形，用于框中其中一架飞机
plane_rect = pygame.Rect(0,99,102,126)

1.3 抠出这架飞机，形成一张单独的图像
plane = plane_img.subsurface(plane_rect)


2. 设置飞机图片的 坐标
screen.blit(plane, (200,500))
3. 刷新窗口屏幕
pygame.display.update()
```

方案1 和 方案2 的区别:

方案1: 采用的方法与 窗口背景一样. 不过需要的图片素材, 都是独立才行

那么后期, 游戏需要的素材会越来越多, 素材的体积占比还是比较大的.

所以**整个游戏**项目的**体积**也会**越大越大**, 我们这才是只是个小游戏而已. 要大游戏还得了

方案2: 飞机大战游戏中, 会有很多飞机, 每个单独的飞机体积并不大, 将各个小飞机 合并在一张图片中.

这样整体体积, 会比一个一个独立存在所占体积小点. 更加的**节省空间**

那么用时, 可以将 大图**一次性加载**过去, 需要哪个小飞机, **抠出来**即可

抠图步骤:

- 通过 `pygame.image.load()` 加载大图
- 通过 `pygame.Rect()` 在大图中创建一个飞机的尺寸矩形
- 通过 `大图.subsurface(小飞机矩形)` 来抠图

抠图分析

pygame.Rect(x, y, width, height)

- 功能
  - 创建一个矩形
- 参数
  - x,y 矩形从哪个坐标 开始创建矩形
  - width, height 矩形有多宽多高
- 返回值
  - Rect矩形对象, 含有x,y坐标, width,height宽高
    - Rect的属性
      - Rect.size 获取矩形大小 元组 (width.height)
      - Rect.width 获取矩形的宽度
      - Rect.height 获取矩形的高度
      - Rect.x 获取矩形的x坐标
      - Rect.y 获取矩形的y坐标

大图.subsurface(小飞机矩形)

- 功能
  - 抠图
- 参数
  - Rect矩形对象
- 返回值:
  - Surface 对象

移动 我方英雄飞机

移动飞机, 主要是通过 **改变**飞机图像的 **坐标**

```
移动 我方英雄飞机
plane_pos[1] -= 30
screen.blit(plane, plane_pos)
pygame.display.update()
```

这里plane_pos 是前面的英雄飞机用坐标的变量. plane_pos = [200, 500]

现在用变量来存储, 主要方便我们移动飞机

要点分析

- plane_pos[1] -= 30
    - 就是操作Y轴, 每次递减30, 也就意味飞机会向上移动30px
    - 若操作X轴, 设置索引为 0 即可
- 设置好坐标后, 需要重新刷新屏幕, 否则无法看到最新的界面

后遗症

发现飞机是移动了, 但是屏幕还**保留**曾经的飞机 **移动的轨迹**

解决方案:

将 屏幕每次操作前 都**重置绘制一次**即可

```
    重新绘制背景  (用于去除原来的残迹)
    screen.blit(background, (0,0))

    移动  我方英雄飞机
    plane_pos[1] -= 30
    screen.blit(plane, plane_pos)
    pygame.display.update()
```

键盘控制飞机

1. 设置飞机 移动的偏移量
2. 获取 按键
3. 计算 飞机移动后的坐标

1. 偏移量

```
offset = {pygame.K_LEFT:0, pygame.K_RIGHT:0, pygame.K_UP:0, pygame.K_DOWN:0}
```

要点分析

这是用于存储 飞机**上下左右** 偏移了多少位置, 方便**重新设置** 飞机坐标

2. 获取按键

```
for event in pygame.event.get():
    按下键盘按键
    if event.type == pygame.KEYDOWN:
        if event.key in offset:
            offset[event.key] = 5

    松开键盘按键
    elif event.type == pygame.KEYUP:
        if event.key in offset:
            offset[event.key] = 0
```

要点分析

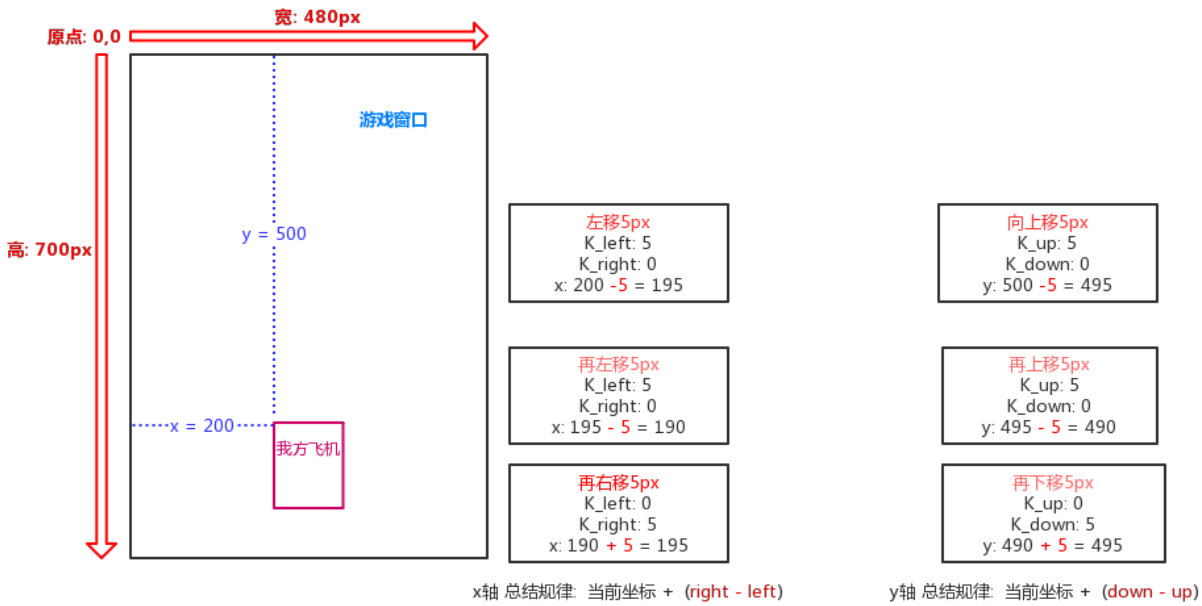
- 按下键盘按键
  - 通过event.key 来获取按键, 如果是offset的成员, 则证明是 方向键之一
  - 无论哪个方向, 都是移动 5位. 当然这个自由调整, 这个其实就是移动速度
- 松开键盘按键
  - 一旦松开, 就意味着不再 挪动飞机了, 那么偏移量设为0

3. 计算移动后的坐标

```
plane_pos[0] = plane_pos[0] + offset[pygame.K_RIGHT] - offset[pygame.K_LEFT]

plane_pos[1] = plane_pos[1] + offset[pygame.K_DOWN] - offset[pygame.K_UP]
```

- plane_pos 存储的是当前飞机的坐标
- plane_pos[0] 是 x轴坐标
- plane_pos[1] 是 y轴坐标

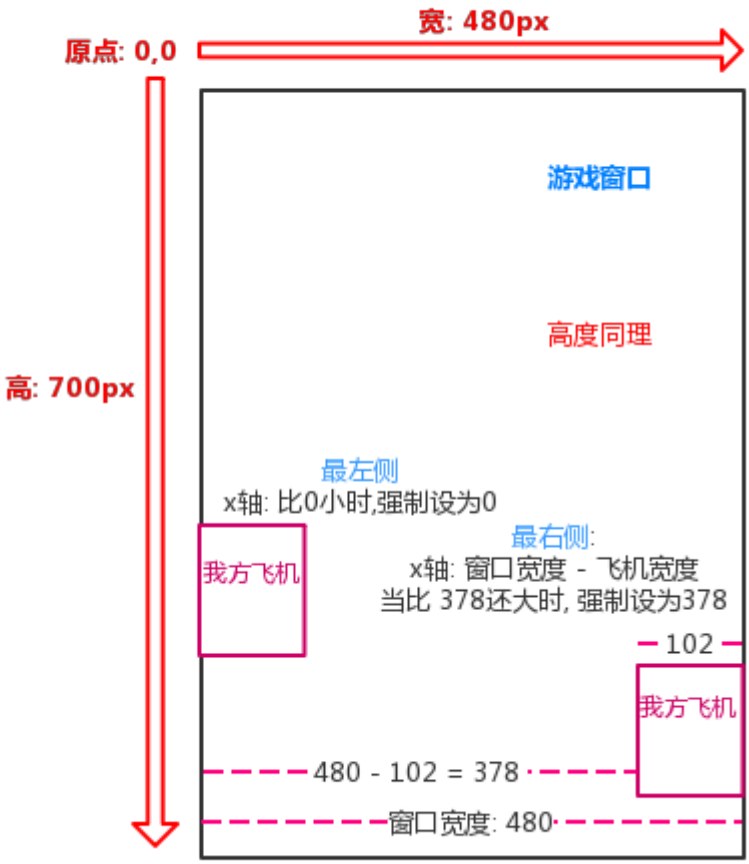


边框限制

```
x轴
if plane_pos[0] < 0:
    plane_pos[0] = 0
elif plane_pos[0] > 480-plane_rect.width:
    plane_pos[0] = 480-plane_rect.width

y轴
if plane_pos[1] < 0:
    plane_pos[1] = 0
elif plane_pos[1] > 700-plane_rect.height:
    plane_pos[1] = 700-plane_rect.height
```

要点分析



帧率

游戏每一个画面都称之为一帧

当画面连续 翻动时, 会给人一种 动起来的感觉, 1s内翻的越多, 发现动画越流畅.

一个**正常人**, 当**1s**内翻**60**次, 感觉会比较**流畅**, 不会有感觉卡顿

Python中, 可以通过 `pygame.time.Clock()` 创建**时钟**, 从而设置**帧率**

```
clock = pygame.time.Clock()
clock.tick(60)
```

要点分析

- pygame.time.Clock() 创建一个时钟
- 时钟.tick(帧率) 在1s 内刷新多少帧

精灵, 精灵组

目前为止, 仔细观察, 会发现, 这个游戏无非就是, **加载图像**, **设置图像坐标**, **绘制图像**

无论是 背景, 还是我方英雄飞机, 甚至将来的敌机, 子弹 等, 都应该是这样的操作

那么每一次都要写这么多代码吗???

这里, **精灵** 和 **精灵组** 就能够**简化**开发步骤

python提供了两个类:

- pygame.sprite.Sprite - 精灵
- pygame.sprite.Group - 精灵组

精灵

每一架敌机, 我方英雄飞机 或者 子弹 都可以理解为 精灵

他们都具备自己的属性 和 方法 (**精灵都是自定义的模板**, 等着别人来**继承**)

案例:

一架敌机, 需要的步骤: 加载敌机图像, 绘制图像, 移动速度 等操作.

那么10架敌机呢? 是不是就要将上面的步骤重复写10次

简化方案:

将一架敌机**继承**精灵, 并补充敌机自己的功能. (精灵本身功能不多)

例如加载图像, 绘制图像, 移动速度 等功能操作

那么以后 需要一架敌机时, 直接**调用精灵**即可, 就不用再写那么多步骤了

1. pygame.sprite.**Sprite** (需要**被继承**)

pygame.sprite.Sprite 精灵类的 几个必备属性 (**Surface对象**, **对象矩形**)

属性	**作用**
image	存放Surface对象
rect	图像矩形的显示位置

> rect 在精灵中, 可通过 **surface.get_rect()** 来获取

pygame.sprite.Sprite 精灵类的 常用方法

方法名	**作用**	**参数**
update()	控制精灵行为	*arg
kill()	从组中 删除精灵	

案例代码

```
plane_role.py 文件

敌机精灵
class Enemy(pygame.sprite.Sprite):

    初始化敌机
    def __init__(self, image_name, speed=1):
        super().__init__()                  调用精灵类的初始化方法(不能重写精灵的初始化)

        设置敌机的属性
        self.image = image_name             加载敌机图像
        self.rect = self.image.get_rect()   获取敌机图像的矩形
```

```
plane.py 文件

准备 敌机图像
enemy_img = pygame.image.load('image/shoot.png')
enemy_rect = pygame.Rect(538,615,52,34)
enemy_small = plane_img.subsurface(enemy_rect)

调用精灵
enemy1 = Enemy(enemy_img)
```

精灵组

精灵组就是 精灵的容器, 方便集体操作.

例如, 有100个精灵同时需要绘制, 按照以往的写法, 需要一个精灵一个精灵的绘制, 代码繁琐, 量大.

而如果把 100个精灵扔进 精灵组. 精灵组只有绘制一次, 那么所有精灵都会各自执行一次.

1. pygame.sprite.**Group**

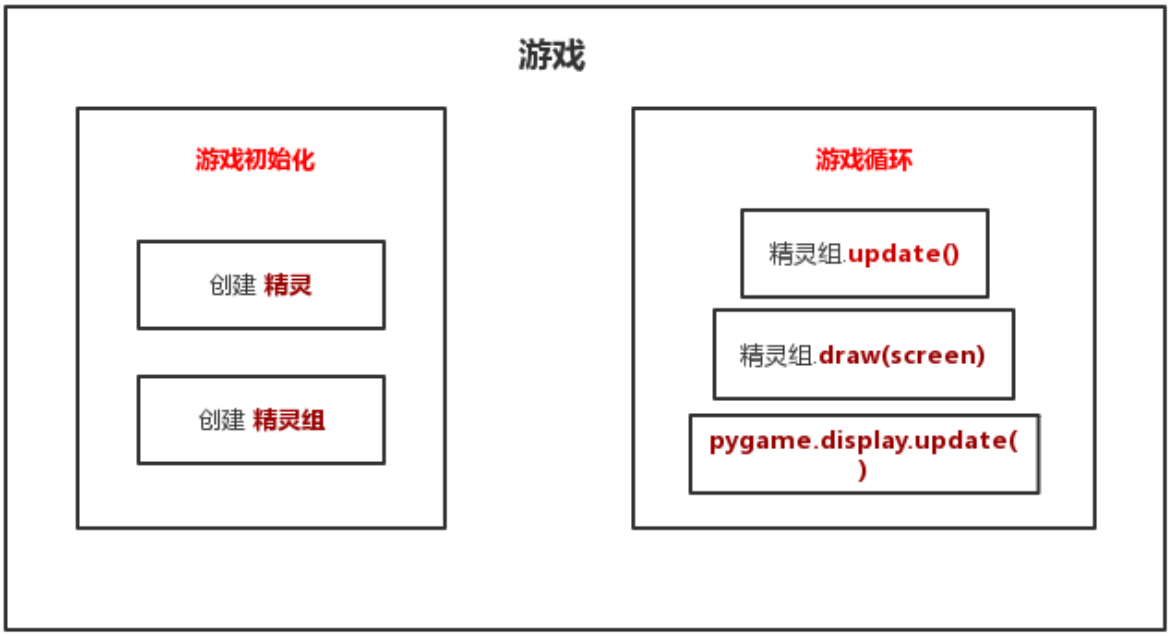
   精灵组的 常用方法

方法名	作用	参数
update()	组中所有精灵都调用 自己update()方法. **就是更新精灵的位置**	*arg
draw()	绘制组中所有的精灵	Surface
add()	向组中添加精灵	
sprites()	返回精灵列表	*sprites

```
enemy_group = pygame.sprite.Group(enemy1)   # 将敌机1 放入精灵组

enemy_group.update()                        # 将精灵组中所有精灵集体调用 update()
enemy_group.draw(screen)                    # 在screen中 绘制所有的精灵
```

整体结构


游戏
游戏初始化
创建 精灵
创建 精灵组
游戏循环
精灵组.update()
精灵组.draw(screen)
pygame.display.update()