

# 游戏框架搭建

目标 —— 使用 面相对象 设计 飞机大战游戏类

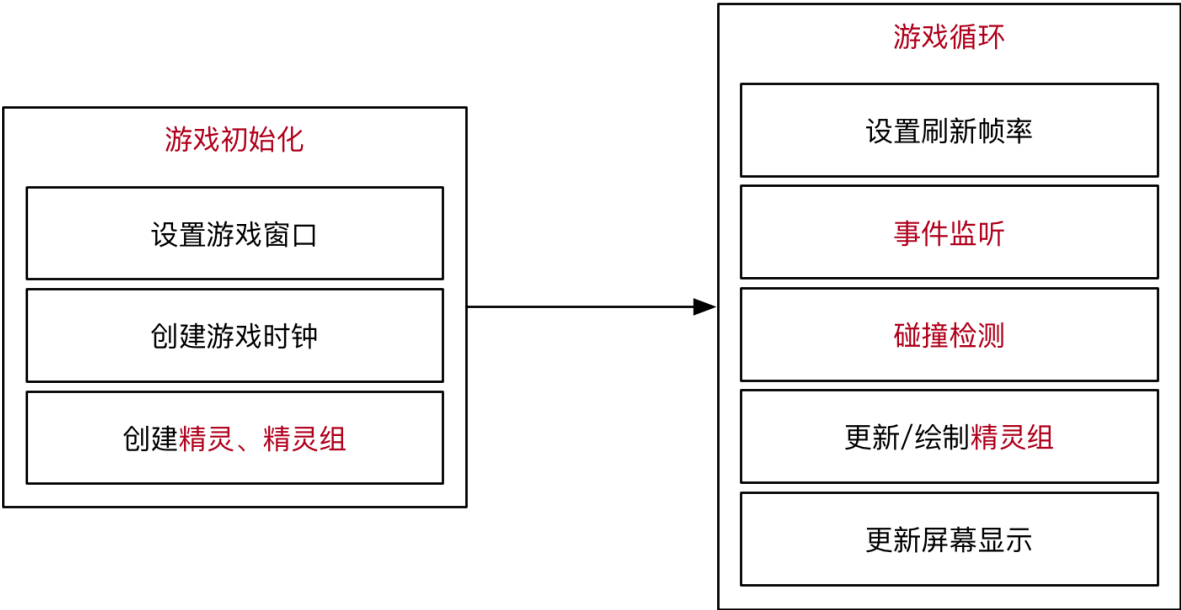
## 目标

- 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：

PlaneGame
<code>screen</code> <code>clock</code> 精灵组或精灵...
<code>__init__(self):</code> <code>__create_sprites(self):</code>  <code>start_game(self):</code> <code>__event_handler(self):</code> <code>__check_collide(self):</code> <code>__update_sprites(self):</code> <code>__game_over():</code>



**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：

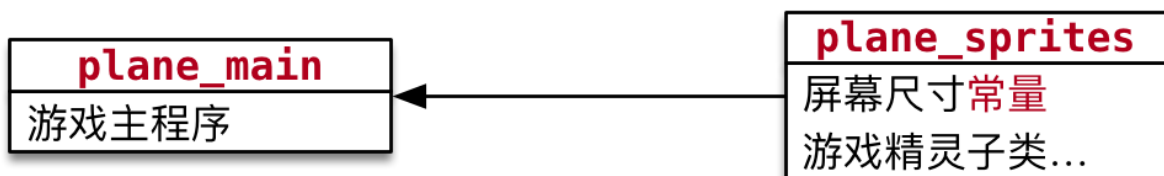
方法	职责
<code>__create_sprites(self)</code>	创建所有精灵和精灵组

- **游戏循环** —— `start_game()` 会调用以下方法：

方法	职责
<code>__event_handler(self)</code>	事件监听
<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



- `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**

```

1  import pygame
2  from plane_sprites import *
3
4
5  class PlaneGame(object):
6      """飞机大战主游戏"""
7
8      def __init__(self):
9          print("游戏初始化")
10
11      def start_game(self):
12          print("开始游戏...")
13
14
15  if __name__ == '__main__':
  
```

```

16     # 创建游戏对象
17     game = PlaneGame()
18
19     # 开始游戏
20     game.start_game()
21

```

## 2.3 游戏初始化部分

- 完成 `__init__()` 代码如下：

```

1  def __init__(self):
2      print("游戏初始化")
3
4      # 1. 创建游戏的窗口
5      self.screen = pygame.display.set_mode((480, 700))
6      # 2. 创建游戏的时钟
7      self.clock = pygame.time.Clock()
8      # 3. 调用私有方法，精灵和精灵组的创建
9      self.__create_sprites()
10
11  def __create_sprites(self):
12      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，开发时不要輕易的修改！

#### 代码调整

- 在 `plane_sprites.py` 中增加常量定义

```

1  import pygame
2
3  # 游戏屏幕大小
4  SCREEN_RECT = pygame.Rect(0, 0, 480, 700)

```

- 修改 `plane_main.py` 中的窗口大小

```
1 self.screen = pygame.display.set_mode(SCREEN_RECT.size)
```

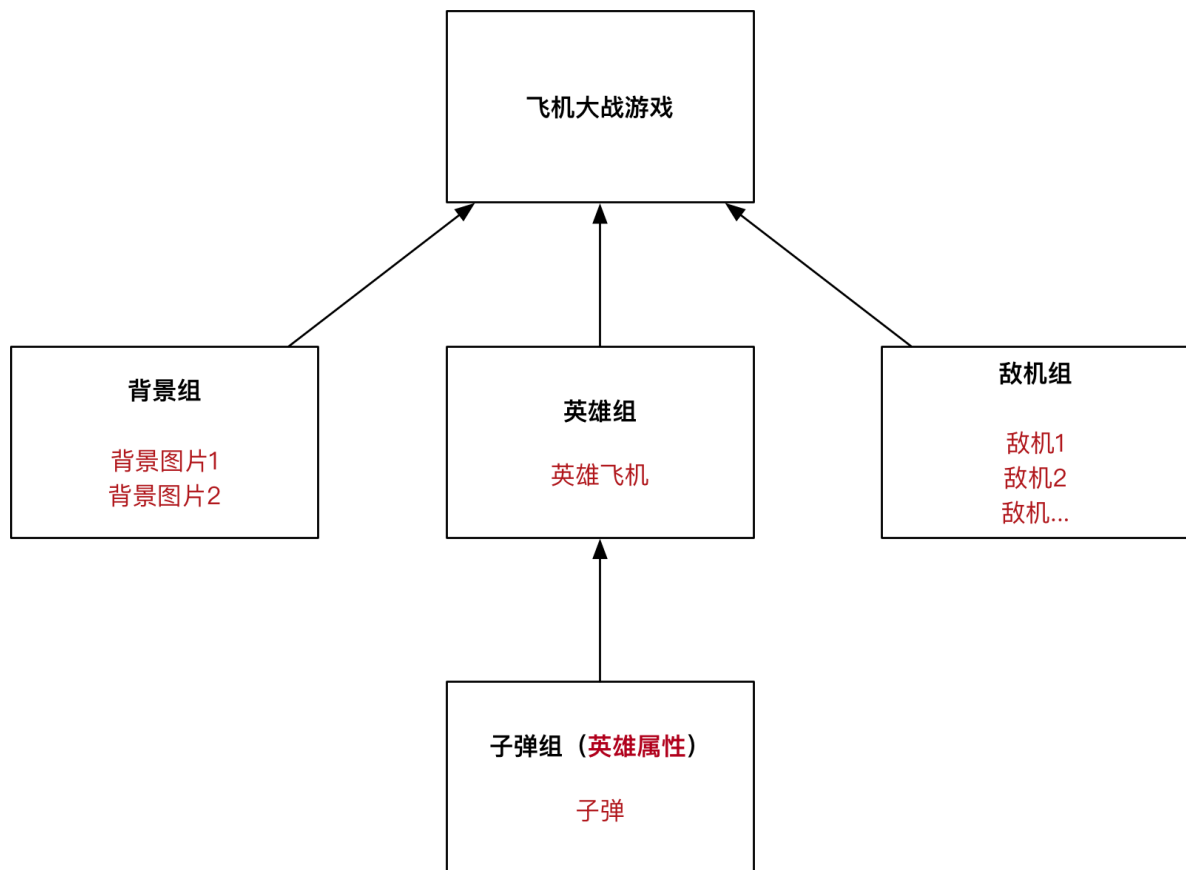
## 2.4 游戏循环部分

- 完成 `start_game()` 基础代码如下：

```
1 def start_game(self):
2     """开始游戏"""
3
4     print("开始游戏...")
5
6     while True:
7
8         # 1. 设置刷新帧率
9         self.clock.tick(60)
10
11        # 2. 事件监听
12        self.__event_handler()
13
14        # 3. 碰撞检测
15        self.__check_collide()
16
17        # 4. 更新精灵组
18        self.__update_sprites()
19
20        # 5. 更新屏幕显示
21        pygame.display.update()
22
23    def __event_handler(self):
24        """事件监听"""
25
26        for event in pygame.event.get():
27
28            if event.type == pygame.QUIT:
29                PlaneGame.__game_over()
30
31    def __check_collide(self):
32        """碰撞检测"""
33        pass
34
35    def __update_sprites(self):
36        """更新精灵组"""
37        pass
38
39    @staticmethod
40    def __game_over():
41        """游戏结束"""
42
43        print("游戏结束")
44        pygame.quit()
45        exit()
```

## 03. 准备游戏精灵组

### 3.1 确定精灵组



## 3.2 代码实现

- 创建精灵组方法

```
1 def __create_sprites(self):
2     """创建精灵组"""
3
4     # 背景组
5     self.back_group = pygame.sprite.Group()
6     # 敌机组
7     self.enemy_group = pygame.sprite.Group()
8     # 英雄组
9     self.hero_group = pygame.sprite.Group()
10
```

- 更新精灵组方法

```
1 def __update_sprites(self):
2     """更新精灵组"""
3
4     for group in [self.back_group, self.enemy_group, self.hero_group]:
5
6         group.update()
7         group.draw(self.screen)
```