项目实战_03_游戏框架搭建

游戏框架搭建 黑马程序员《Python 入门教程完整版》笔记

一 黑马程序员《Python 入门教程完整版》笔记

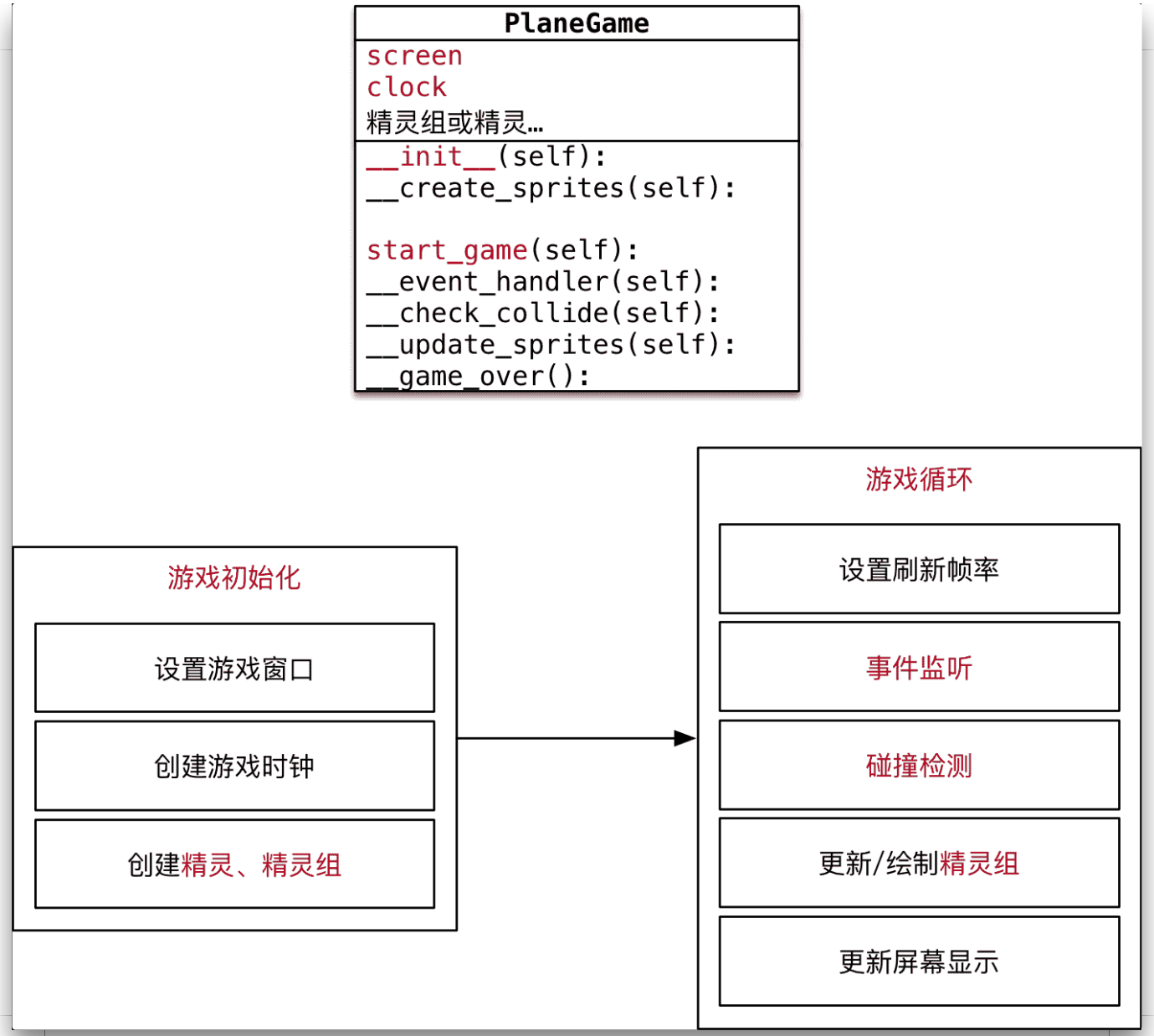
目标 —— 使用 面相对象 设计 飞机大战游戏类

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：



> **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

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**

```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")

    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：

```python
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

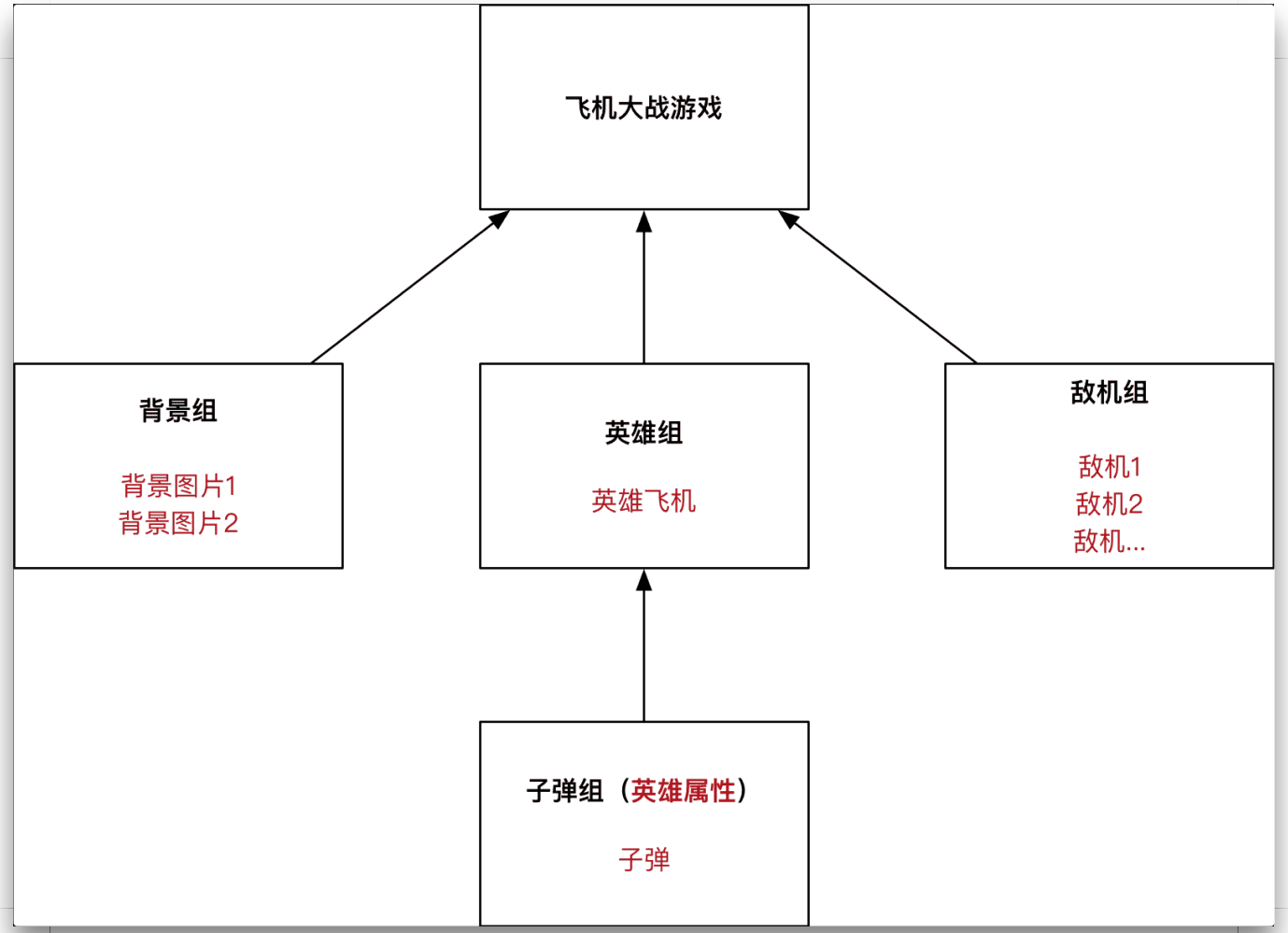
- 完成 `start_game()` 基础代码如下：

```
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



3.2 代码实现

- 创建精灵组方法

```python
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

```python
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)
```