项目名称：基于 opencv 色彩追踪的飞机大战

小组名称：Debuggers

组长：王泳茗

组员：方恒彦，周晟卿

指导老师：鲍杨

目录

一、 分工：

王泳茗：决定选题，查找资料，编写与修改游戏主体的代码，后期的调试，写项目报告中游戏规则与代码概述中的游戏部分。
方恒彦：决定选题，查找资料，色彩追踪部分代码与调试，写项目报告中选题背景、代码概述中的色彩追踪部分、总结，准备 demo。
周晟卿：决定选题，查找资料，代码追踪与游戏结合的代码，调试，写项目报告中使用的库、代码概述中的追踪与游戏主循环的结合、展望，准备 demo。

二、 选题背景：

飞机大战是我们最早接触的经典游戏之一。在本项目中，我们试图创新游戏机制，脱离键盘来操纵这款游戏。我们自学 opencv 和 pygame，熟悉库中的各类语法操作。通过 opencv 我们实现了实时追踪红色物体的功能，通过 pygame 我们创建了几个模块实现飞机大战游戏的各个功能。经过代码整合与调试后，我们只需要打开摄像头，并手持红色物体，通过改变手中红色物体的位置，就控制飞船的左右，击打目标物体。

三、 游戏规则：

运行 run_the_game.py 进入游戏，打开摄像头，点击 start 按钮游戏开始。界面中会有一定数量的外星人，从屏幕上方袭来。他们在界面中依次左右移动，当移动到屏幕边缘，就会向下移动一格。我们要做的是通过手中红色物体的位置控制飞船左右移动，而飞船会自动开火击落外星人。如果有外星人撞击到我们的飞船，或有外星人到达了屏幕最下方，则本轮游戏失败，我们减少一架飞船。 如果界面中所有外星人均被击落，则本轮游戏胜利，等级加一，会有新的一批外星人以更快的速度袭来。我们共有三架飞船，即三条性命，当我们没有剩下的飞船时，游戏结束，达到的等级为最终成绩。

四、 使用的库

1、Pygame

Pygame 是一个免费，开源的编程语言库，里面提供了使用 Python 开发游戏的基础包。它建立在 SDL 基础上，允许实时电子游戏研发而无需被低级语言（如机器语言和汇编语言）束缚，Pygame 具有高度的可移植性，几乎可以在所有平台和操作系统上运行。
我们基于 Pygame 完成了飞机大战游戏的代码。

2、OpenCv

OpenCv 是一个基于 BSD 许可发行的免费，开源跨平台计算机视觉库。它用 C++语言编写，同时提供了 Python, Ruby, MATLAB 等语言的接口，实现了图像处理和计算机视觉方面的很多通用算法。
我们基于 OpenCv 完成了对红色物体追踪识别的代码，运用 OpenCv 进行了 rgb 至 hsv 的转换，腐蚀，轮廓检测，计算质心等操作。

3、Numpy

NumPy 是 Python 的一种开源的数值计算扩展。这种工具可用来存储和处理大型矩阵，比 Python 自身的嵌套列表结构要高效的多，支持大量的维度数组与矩阵运算，此外也针对数组运算提供大量的数学函数库。

Numpy 在该项目中运用不多，我们运用它设定了一个三阶数组，以来为红色的 hsv 阈值筛选做准备。

五、 代码概述

1、色彩追踪

```
26      #设定红色阈值，HSV空间
27      redLower = np.array([170, 100, 100])
28      redUpper = np.array([179, 255, 255])
29
30      #打开摄像头
31      camera = cv2.VideoCapture(0)
```

首先，我们通过 np.array 创建数组，设定红色的阈值。我们在 HSV 空间设定所识别的红色物体的 HSV 上限与下限，以便在后续色彩识别中使用。相对于 RGB 空间，HSV 空间能够非常直观的表达色彩的明暗，色调，以及鲜艳程度，方便进行颜色之间的对比。接着用 cv2.VedioCapture(0)打开系统默认的摄像头。

```
68      # Start the main loop for the game.
69      while True:
70          (ret, frame) = camera.read()
71          frame = cv2.flip(frame, 1)  # 图片翻转   Y轴翻转
72          #判断是否成功打开摄像头
73          if not ret:
74              print('No Camera')
75              break
```

我们通过 camera.read()按帧读取视频，ret 获取布尔值，如果读取帧是正确的则返回 True，如果文件读取到结尾，它的返回值就为 False。frame 获取每一帧的图像，是个三维矩阵。下面通过 cv2.flip 对每帧的图像进行水平翻转，从而使飞机移动方向与红色物体移动方向相一致。为确保摄像头运行正常，加入条件语句进行判断，及时终止循环。

```
        #转到HSV空间
        hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
        #根据阈值构建掩膜
        mask = cv2.inRange(hsv, redLower, redUpper)
        #腐蚀操作
        mask = cv2.erode(mask, None, iterations=2)
        #膨胀操作，其实先腐蚀再膨胀的效果是开运算，去除噪点
        mask = cv2.dilate(mask, None, iterations=2)
        #轮廓检测
        cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)[-2]
```

接下来进行图像处理。首先将 RGB 图像转化为 HSV 图像。然后利用 cv2.inRange 函数设阈值，结合之前定义的红色上限下限，去除背景部分。再利用 cv2.erode 进行图像腐蚀，cv2.dilate 进行图像膨胀，去除噪点。最后利用 cv2.findContours 进行轮廓检测，cv2.RETR_EXTERNAL 表示只检测外轮廓，cv2.CHAIN_APPROX_SIMPLE 压缩水平方向，垂直方向，对角线方向的元素，只保留该方向的终点坐标。最终返回轮廓 cnts。

```
            #初始化瓶盖圆形轮廓质心
            center = None
            #如果存在轮廓
            if len(cnts) > 0:
                #找到面积最大的轮廓
                c = max(cnts, key = cv2.contourArea)
                #计算轮廓的矩
                M = cv2.moments(c)
                #计算质心
                center = (int(M["m10"]/M["m00"]), int(M["m01"]/M["m00"]))
```

下面进行坐标处理。首先定义轮廓中心 center。接着取 cnts 面积最大的轮廓为 c，再运用函数 cv2.moments()将计算得到的矩以一个字典的形式返回。根据这些矩的值，我们可以计算出对象的中心 cx=int(M['m10']/M['m00'])，cy=int(M['m01']/M['m00'])。

2、追踪与游戏主循环的结合

```
                if int(M["m10"]/M["m00"]) >360 :
                    ship.moving_right = True
                    ship.moving_left = False
                elif int(M["m10"]/M["m00"]) <280 :
                    ship.moving_right = False
                    ship.moving_left = True
                elif int(M["m10"]/M["m00"]) >=280 and int(M["m01"]/M["m00"]) <=360 :
                    ship.moving_right = False
                    ship.moving_left = False

```

由于我们在追踪红色物体中获得了红色物体质心的坐标，也就表明了这个坐标的数值可以作为一个条件来控制飞船的移动。为了让我们的操作与飞船的左右移动相吻合，我们用质心横坐标的数值来将追踪与游戏主循环相结合。

由于摄像头捕捉到的画面横坐标范围是 0 至 640，我们将此范围分成三个部分，分别是

0 ~ 280，280 ~ 360，360 ~ 640，即左半部分，中间部分，右半部分，分别对应的是飞船向左移动，不动，向右移动。

这样，我们只需移动手中的红色物体就可控制飞船了，红色物体的位置就代表了飞船的运动方向，放在左边飞船向左移动，放在中间则飞船不动，放在右边飞船向右移动。

3、游戏所用对象

我们创建了几个模块，包含不同的类，来管理游戏中不同的对象。我们将这些模块的.py 文件放在游戏主体代码 run_the_game.py 的同一目录下。这样以来，在游戏的主体代码中只要调用相关函数和参数，可以让游戏主体的代码更加简洁，也方便我们的修改。我们共有 settings（基本设置），ship（飞船），bullet（子弹），alien（外星人），button（satrt 按钮），game_stats（游戏数据），scoreborad（计分牌）和 game_functions（游戏功能）八个辅助模块。其中，前七个模块都是一些基本的参数和函数定义，而 game_functions 模块通过调用这些模块，定义了更复杂的函数，在主程序中，我们所用的函数均来自 game_functions 模块。模块（.py 文件）名为小写，它们分别包含的类首字母大写。

3.1 settings 模块

```python
class Settings():
    """存储所有设置参数"""

    def __init__(self):
        """初始化游戏参数"""
        #屏幕参数
        self.screen_width = 1200
        self.screen_height = 800
        self.bg_color = (230, 230, 230)

        #飞船参数
        self.ship_speed_factor = 6
        self.ship_limit = 2

        #子弹参数
        self.bullet_speed_factor = 30
        self.bullet_width = 4
        self.bullet_height = 20
        self.bullet_color = 165, 42, 42
        self.bullets_allowed = 2

        #外星人参数
        self.fleet_direction = 1
        #外星人的速度放在下列函数中
        self.initialize_dynamic_settings()

        #每关中外星人速度变快的程度
        self.speedup_scale = 1.1

    def initialize_dynamic_settings(self):
        """初始化外星人速度，因为会随游戏进行而变化"""
        self.alien_speed_factor = 4
        self.fleet_drop_speed = 30

    def increase_speed(self):
        """提高速度设置"""
        self.alien_speed_factor *= self.speedup_scale
        self.fleet_drop_speed *= self.speedup_scale
```

settings 模块中包含 Settings 类，用来存储屏幕、飞船、外星人、子弹等的参数。其中两个函数分别为初始化外星人的速度和在每关中提高外星人的速度。

3.2 ship 模块

```python
import pygame
from pygame.sprite import Sprite

class Ship(Sprite):

    def __init__(self, ai_settings, screen):
        """初始化飞船"""
        super(Ship, self).__init__()
        self.screen = screen
        self.ai_settings = ai_settings

        #载入飞船图像，获得其矩形对象
        self.image = pygame.image.load('images/ship.bmp')
        self.rect = self.image.get_rect()
        self.screen_rect = screen.get_rect()

        #把飞船放在屏幕最下方居中
        self.rect.centerx = self.screen_rect.centerx
        self.rect.bottom = self.screen_rect.bottom

        #获得矩形对象中点的小数值
        self.center = float(self.rect.centerx)

        #开始时设定移动为False
        self.moving_right = False
        self.moving_left = False

    def update(self):
        """更新飞船的位置"""
        #在飞船不到屏幕边缘时更新飞船横坐标
        if self.moving_right and self.rect.right < self.screen_rect.right:
            self.center += self.ai_settings.ship_speed_factor
        if self.moving_left and self.rect.left > 0:
            self.center -= self.ai_settings.ship_speed_factor

        #把小数值赋给矩形对象
        self.rect.centerx = self.center

    def blitme(self):
        """在当下位置绘制飞船"""
        self.screen.blit(self.image, self.rect)

    def center_ship(self):
        """让飞船在屏幕上居中"""
        self.center = self.screen_rect.centerx
```

ship 模块管理飞船的大部分行为。这里的 ai_settings 会在主体程序中先出现，即有 ai_settings = Settings()，因此 settings 中所存储的飞船速度被导入到 ship 的函数中。其中 update 函数用来更新飞船位置，blitme 函数用来绘制飞船。在游戏主体中，我们要先后调用这两个函数使飞船最新位置显现在屏幕上。这里 Ship 类继承了从 pygame.sprite 导入的 Sprite 类，可以通过调用 Sprite 中的函数 super()来为飞船编组。在操控游戏时，我们只有

一个飞船需要控制，但左上角会显示剩余飞船数，这里的编组也是针对左上角飞船的。

3.3 bullet 模块

```python
import pygame
from pygame.sprite import Sprite

class Bullet(Sprite):
    """管理子弹的类"""
    #继承了Sprite类，可以将子弹编组

    def __init__(self, ai_settings, screen, ship):
        """在飞船所处位置创建一个子弹对象"""
        super(Bullet, self).__init__()
        self.screen = screen

        #创建矩形作为子弹，移到飞船的位置
        self.rect = pygame.Rect(0, 0, ai_settings.bullet_width,
            ai_settings.bullet_height)
        self.rect.centerx = ship.rect.centerx
        self.rect.top = ship.rect.top

        #存储子弹的纵坐标为小数值
        self.y = float(self.rect.y)
        #从settings中得到子弹的颜色和速度
        self.color = ai_settings.bullet_color
        self.speed_factor = ai_settings.bullet_speed_factor

    def update(self):
        """更新子弹位置，即向上移动子弹"""
        #更新子弹纵坐标
        self.y -= self.speed_factor
        self.rect.y = self.y

    def draw_bullet(self):
        """在屏幕上绘制子弹"""
        pygame.draw.rect(self.screen, self.color, self.rect)
```

Bullet 类同样继承了 Sprite，将所有子弹编组。在主程序中，我们导入 pygame.sprite 中的 Group 类，即可用 bullets=Group()，创建一个用于存储子弹的编组。游戏中的子弹是我们创建的矩形，我们定义其大小和位置，获得其纵坐标值。由于子弹垂直向上射出，因此我们不需要修改其横坐标。这里有 update 和 draw_bullet 两个函数，用来更新其纵坐标，和绘制最新的位置。

3.4 alien 模块

```python
import pygame
from pygame.sprite import Sprite

class Alien(Sprite):
    """管理外星人的类"""

    def __init__(self, ai_settings, screen):
        """创建一个卫星人，设置其位置"""
        super(Alien, self).__init__()
        self.screen = screen
        self.ai_settings = ai_settings

        #导入外星人图像，得到其矩形对象
        self.image = pygame.image.load('images/alien.bmp')
        self.rect = self.image.get_rect()

        #把新的外星人放到屏幕左上侧
        self.rect.x = self.rect.width
        self.rect.y = self.rect.height

        #存储外星人的横坐标
        self.x = float(self.rect.x)

    def check_edges(self):
        """检测外星人是否到屏幕左右边缘，如果是，返回True"""
        screen_rect = self.screen.get_rect()
        if self.rect.right >= screen_rect.right:
            return True
        elif self.rect.left <= 0:
            return True

    def update(self):
        """左右移动外星人"""
        self.x += (self.ai_settings.alien_speed_factor *
                       self.ai_settings.fleet_direction)
        self.rect.x = self.x

    def blitme(self):
        """在其实时位置绘制外星人"""
        self.screen.blit(self.image, self.rect)
```

alien 模块与 bullet 模块即为相似，都是继承了 Sprite 类，为外星人编组，在主程序中创建存储外星人的编组也是用同样的方法。我们导入外星人的图像，调用 settings 中的参数，我们这里只需要创建在屏幕左上角的一个外星人，创建多个外星人的函数会在之后的 game_function 中出现。check_edges 函数用来检测外星人是否碰到边缘，因为外星人会平行移动，碰到边缘后向下移动一段距离并反方向平行移动。update 函数中的 self.ai_settings.fleet_direction = -1，用来让外星人获得反方向的不变速度。

3.5 button 模块

```python
import pygame.font

class Button():

    def __init__(self, ai_settings, screen, msg):
        """初始化按钮的属性"""
        self.screen = screen
        self.screen_rect = screen.get_rect()

        #设置按钮和文字的颜色大小
        self.width, self.height = 200, 50
        self.button_color = (100, 155, 0)
        self.text_color = (255, 255, 255)
        self.font = pygame.font.SysFont(None, 48)

        #创建按钮的矩形对象并居中
        self.rect = pygame.Rect(0, 0, self.width, self.height)
        self.rect.center = self.screen_rect.center

        #调用下面的函数渲染文字
        self.prep_msg(msg)

    def prep_msg(self, msg):
        """把文字渲染为图像并使其在按钮上居中"""
        self.msg_image = self.font.render(msg, True, self.text_color,
            self.button_color)
        self.msg_image_rect = self.msg_image.get_rect()
        self.msg_image_rect.center = self.rect.center

    def draw_button(self):
        """绘制按钮与文字"""
        self.screen.fill(self.button_color, self.rect)
        self.screen.blit(self.msg_image, self.msg_image_rect)
```

button 模块针对的是游戏界面中的 start 按钮，这里只有按钮的创建、渲染和绘制，点击按钮的函数会在 game_functions 中。这里引入了 pygame.font，用来创建文字，msg 是代表文字的形参。这里我们先创建了一个矩形框，然后创建文字并将文字渲染为图像，再放置到矩形框的中央。draw_button 函数同时绘制按钮与文字图像于屏幕上。

3.6 game_stats 模块

```python
class GameStats():
    """管理游戏中的一些数据"""

    def __init__(self, ai_settings):
        """初始化数据"""
        self.ai_settings = ai_settings
        self.reset_stats()

        # 让游戏一开始处于非活动状态
        self.game_active = False

    def reset_stats(self):
        """重新初始化数据"""
        self.ships_left = self.ai_settings.ship_limit
        self.level = 1
```

game_stats 模块用来初始化游戏数据，即等级（打到第几关）和剩余飞船，定义了 reser_stats 函数来重新初始化数据。初始等级为 1 级，其中 ship_limit 在 ai_settings 中赋值，我们设置为 2.

3.7 scoreboard 模块

```python
import pygame.font
from pygame.sprite import Group
from ship import Ship

class Scoreboard():
    """管理游戏中的计分"""

    def __init__(self, ai_settings, screen, stats):
        """初始化计分涉及的属性"""
        self.screen = screen
        self.screen_rect = screen.get_rect()
        self.ai_settings = ai_settings
        self.stats = stats

        #显示分数的文字颜色及大小
        self.text_color = (30, 30, 30)
        self.font = pygame.font.SysFont(None, 48)

        #准备初始计分的图像
        self.prep_level()
        self.prep_ships()

    def prep_level(self):
        """把等级渲染为图像"""
        self.level_image = self.font.render(str(self.stats.level), True,
                self.text_color, self.ai_settings.bg_color)

        #设定图像在右上角
        self.level_rect = self.level_image.get_rect()
        self.level_rect.right = self.screen_rect.right - 20
        self.level_rect.top = 20

    def prep_ships(self):
        """把剩余飞船编组定义位置"""
        self.ships = Group() #创建空编组存储飞船
        for ship_number in range(self.stats.ships_left):
            ship = Ship(self.ai_settings, self.screen)
            ship.rect.x = 10 + ship_number * ship.rect.width
            ship.rect.y = 10
            self.ships.add(ship)

    def show_score(self):
        """把计分的图像绘制在屏幕上"""
        #绘制等级
        self.screen.blit(self.level_image, self.level_rect)
        #绘制剩余飞船
        self.ships.draw(self.screen)
```

scoreboard 模块通过调用 game_stats 中的数据，把数据显示在屏幕上。其中 prep_level 函数将等级的数字渲染为图像，放在屏幕右上角。prep_ships 为飞船编组，并给编组中的每个飞船确定了位置。而 show_score 函数基于以上的设置，将等级图像与剩余飞船图像绘制

在屏幕上，对编组调用 draw()可以绘制每艘飞船。

3.8 game_functions 模块

由于这一模块中函数较多，我们将分别解释模块中的函数。

3.8.1 外星人群的创建与方向控制

```
def create_fleet(ai_settings, screen, ship, aliens):
    """创建外星人群"""
    #创建一个外星人，并计算一行可以容纳多少个外星人
    #外星人间距为外星人宽度
    alien = Alien(ai_settings, screen)
    alien_width = alien.rect.width
    alien_height = alien.rect.height
    ship_height = ship.rect.height


    #计算可容纳多少行外星人
    available_space_y = ai_settings.screen_height - 3*alien_height - ship_height
    number_rows = int(available_space_y / (2*alien_height))

    #计算一行可容纳多少个外星人
    available_space_x = ai_settings.screen_width - 2*alien_width
    number_aliens_x = int(available_space_x / (2*alien_width))

    #逐行创建外星人
    for row_number in range(number_rows ):
        for alien_number in range(number_aliens_x ):
            #创建一个外星人并将其加入当前行
            alien = Alien(ai_settings, screen)
            alien.x = 2*alien_width + 2*alien_width*alien_number
            alien.rect.x = alien.x
            alien.rect.y = alien.rect.height + 2*alien.rect.height*row_number
            aliens.add(alien)


def check_fleet_edges(ai_settings,aliens):
    """外星人到屏幕边缘后的操作"""
    for alien in aliens.sprites():
        if alien.check_edges():
            change_fleet_direction(ai_settings, aliens)#调用先下移转向的函数
            break

def change_fleet_direction(ai_settings,aliens):
    """将整群外星人下移，并改变它们的方向"""
    for alien in aliens.sprites():
        alien.rect.y += ai_settings.fleet_drop_speed#下移
    ai_settings.fleet_direction = -1*ai_settings.fleet_direction#横方向转向
```

定义了 create_fleet 函数，来创建一群外星人。我们根据屏幕大小，外星人大小来确定

外星人的排列方式，然后逐行创建外星人，定义他们的位置，并加到编组中。change_fleet_direction 将所有外星人下移，并横方向转向。check_fleet_edges 函数是它的调用条件，即有一个外星人到屏幕边缘，所有外星人执行下移转向。

3.8.2 本轮游戏失利的情况

```
146  def ship_hit(ai_settings, stats, screen, sb, ship, aliens, bullets):
147      """飞船被外星人撞到后的操作"""
148      if stats.ships_left > 0:
149          stats.ships_left -= 1
150          #更新记分牌中剩余飞船的个数
151          sb.prep_ships()
152          #清空外星人列表和子弹列表
153          aliens.empty()
154          bullets.empty()
155
156          #创建一群新的外星人，飞船回屏幕中央
157          create_fleet(ai_settings, screen, ship, aliens)
158          ship.center_ship()
159
160          #暂停
161          sleep(0.5)
162      else: #如果没有剩余飞船，则游戏结束
163          stats.game_active = False
164          pygame.mouse.set_visible(True)
165
166  def check_aliens_bottom(ai_settings, stats, screen, sb, ship, aliens, bullets):
167      """检测外星人是否到底部，并执行相关操作"""
168      screen_rect = screen.get_rect()
169      for alien in aliens.sprites():
170          if alien.rect.bottom >= screen_rect.bottom:
171              #外星人到底部的结果和飞船被撞到一样，故调用ship_hit函数
172              ship_hit(ai_settings, stats, screen, sb, ship, aliens, bullets)
173              break
```

如果外星人撞到飞船或到达屏幕底部，本轮游戏失利。这里的 ship_hit 函数定义了飞船被撞的操作，如果剩余飞船大于零，则剩余飞船-1，并清空本轮的外星人和子弹编组，调用 create_fleet 函数重新创建外星人群。如果剩余飞船为零，则 stats.game_active 为 False，游戏结束。而 check_aliens_bottom 函数通过检测每个外星人的底部坐标，如果到达屏幕低端，则执行 ship_hit 函数，因为两种情况的结果是一样的。

3.8.3 更新外星人位置（外星人相关函数的综合）

```
def update_aliens(ai_settings, stats, screen, sb, ship, aliens, bullets):
    """检测外星人是否到屏幕边缘，更新外星人群中所有外星人的位置"""
    check_fleet_edges(ai_settings,aliens)
    aliens.update()

    #检测外星人与飞船之间的碰撞
    if pygame.sprite.spritecollideany(ship, aliens):
        ship_hit(ai_settings, stats, screen, sb, ship, aliens, bullets)

    check_aliens_bottom(ai_settings, stats, screen, sb, ship, aliens, bullets)
```

update_aliens 的函数用来更新所有外星人的位置。其中调用 check_fleet_edges 函数执行边缘检测，调用 aliens.update 函数更新外星人编组。为 ship_hit 函数添加了条件，pygame.sprite.spritecollideany 检测飞船和外星人的矩形对象是否重合，若重合，则执行 ship_hit 的结果。调用 check_aliens_bottom 函数检测外星人是否到底部并执行相关操作。update_aliens 函数是对与外星人有关函数的综合，在主程序的主循环中，我们只需要调用这一函数，就可以不断地检测和控制外星人。

3.8.4 射击与子弹更新

```
def fire_bullet(ai_settings, screen, ship, bullets):
    """射出子弹"""
    #创建新子弹，将子弹加入编组中
    if len(bullets) < ai_settings.bullets_allowed:
        new_bullet = Bullet(ai_settings, screen, ship)
        bullets.add(new_bullet)

def update_bullets(ai_settings, screen, stats, sb, ship, aliens, bullets):
    """更新子弹位置，删掉多余的子弹，执行击落外星人的操作，更新等级，创建新外星人群"""
    #更新子弹的位置
    bullets.update()

    #从子弹编组中删掉已消失的子弹
    for bullet in bullets.copy():
        if bullet.rect.bottom <= 0:
            bullets.remove(bullet)

    #如果子弹击中了外星人，就删除子弹与外星人
    collisions = pygame.sprite.groupcollide(bullets, aliens, True, True)

    if len(aliens) == 0:#即玩家击落了这一轮全部外星人
        #删除现有的子弹并创建新的一群外星人
        bullets.empty()
        ai_settings.increase_speed() #使新外星人移动速度加快
        stats.level += 1 #等级加一
        sb.prep_level() #将等级绘制成图像渲染出
        create_fleet(ai_settings, screen, ship, aliens)
```

fire_bullet 函数用来创建子弹，如果现有的子弹在允许数值范围内，则创建新子弹加入编组。Update_bullet 有多个功能，一，调用 bullet 模块中的 update 函数来更新编组中子弹的位置。二，删除 bullets 编组中屏幕外的子弹，即让 bullets 编组中永远只存储屏幕内的子

弹，加快程序的速度。三，调用 pygame.sprite.groupcollide 函数，来实现击落外星人。如果外星人的矩形对象和子弹重叠，则删除外星人与子弹。四，如果本轮屏幕中所有外星人均被击落，则清空子弹编组，加快新一轮外星人的速度，提高等级和创建新的外星人群。

3.8.5 检测鼠标操作

```
def check_events(ai_settings, screen, stats, sb, play_button, ship, aliens, bullets):
    """检测鼠标操作"""
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            sys.exit()
        elif event.type == pygame.MOUSEBUTTONDOWN:
            #如果单击start按钮，调用开始新游戏的函数
            mouse_x, mouse_y = pygame.mouse.get_pos()
            check_play_button(ai_settings, screen, stats, sb, play_button, ship, aliens, bullets, mouse_x,mouse_y)


def check_play_button(ai_settings, screen, stats, sb, play_button, ship, aliens, bullets, mouse_x,mouse_y):
    """在玩家单击start按钮时开始新游戏"""
    if play_button.rect.collidepoint(mouse_x, mouse_y) and not stats.game_active:

        #重置外星人速度
        ai_settings.initialize_dynamic_settings()

        #隐藏光标
        pygame.mouse.set_visible(False)

        #重置游戏统计信息
        stats.reset_stats()
        stats.game_active = True

        #重置记分牌图像
        sb.prep_level()
        sb.prep_ships()
```

虽然我们的游戏不需要键盘，但要鼠标点击 start 按钮以开始游戏，用鼠标关闭窗口以退出游戏界面。check_events 函数中，用 pygame.Quit 检测是否点击关闭按钮，sys.exit()用以退出游戏。Pygame.MOUSEBUTTONDOWN 检测鼠标点击的操作，我们获得点击时鼠标的坐标，并调用 check_play_button 函数来执行点击 start 按钮的操作。如果鼠标坐标与 start 按钮重合，则执行初始化外星人速度，隐藏鼠标，初始化数据和记分牌，stats.game_active = True，即可开始游戏。

3.8.6 更新屏幕的绘制

```
def update_screen(ai_settings, stats, screen, sb, ship, aliens, bullets, play_button):
    """更新屏幕，绘制新内容"""
    #重画一遍背景
    screen.fill(ai_settings.bg_color)

    #更新子弹，飞船和外星人
    for bullet in bullets.sprites():
        bullet.draw_bullet()
    ship.blitme()
    aliens.draw(screen)

    #显示相关关卡数和剩余飞船
    sb.show_score()

    #如果游戏处于非活动状态，就绘制start按钮
    if not stats.game_active:
        play_button.draw_button()

    #让最新绘制的屏幕可见
    pygame.display.flip()
```

之前更新的函数 update_bullets 和 update_aliens 只是执行了相关操作，改变了对象的位置，但并没有绘制显现到屏幕上。这里定义 uodtae_screen 函数，来达到这一目的。Fill 函数用来填充屏幕每次循环都会重新绘制背景；对于子弹，调用 bullet 模块中 draw_bullet 函数以绘制；同样的 ship.blitme()，aliens.draw(screen)，sb.show_score()都会重绘相关对象。对于 start 按钮，如果游戏非活动，则绘制。最后，用 pygame.display.flip()让最新绘制的整个屏幕可见。显然，update_screen 要放到循环的最后。

4、游戏主程序

```
    #初始化pygame、设置和屏幕
    pygame.init()
    ai_settings = Settings()
    screen = pygame.display.set_mode(
        (ai_settings.screen_width, ai_settings.screen_height))
    pygame.display.set_caption("Alien Invasion")
    #创建Start按钮
    play_button = Button(ai_settings, screen, "START")
    #创建一个用于存储游戏统计信息的实例
    stats = GameStats(ai_settings)
    #创建记分牌
    sb = Scoreboard(ai_settings, screen, stats)
    #创建背景颜色
    bg_color = (230, 230, 230)
    #创建飞船
    ship = Ship(ai_settings, screen)
    #创建存储子弹的编组
    bullets = Group()
    #创建存储外星人的编组
    aliens = Group()
    #创建外星人群
    gf.create_fleet(ai_settings, screen, ship, aliens)
```

在游戏主程序 run_the_game 中，我们先带入必要的库与此前创建的模块。在主循环之

前，先用 pygame.init()进行初始化，然后创建一系列的实例，存储在相关变量中。之后我们使用函数时都会将这些变量作为实参进行传递。此前我们导入 game_functions 时赋予其 gf 的名称，因此这里调用 game_function 的函数都用 gf.()

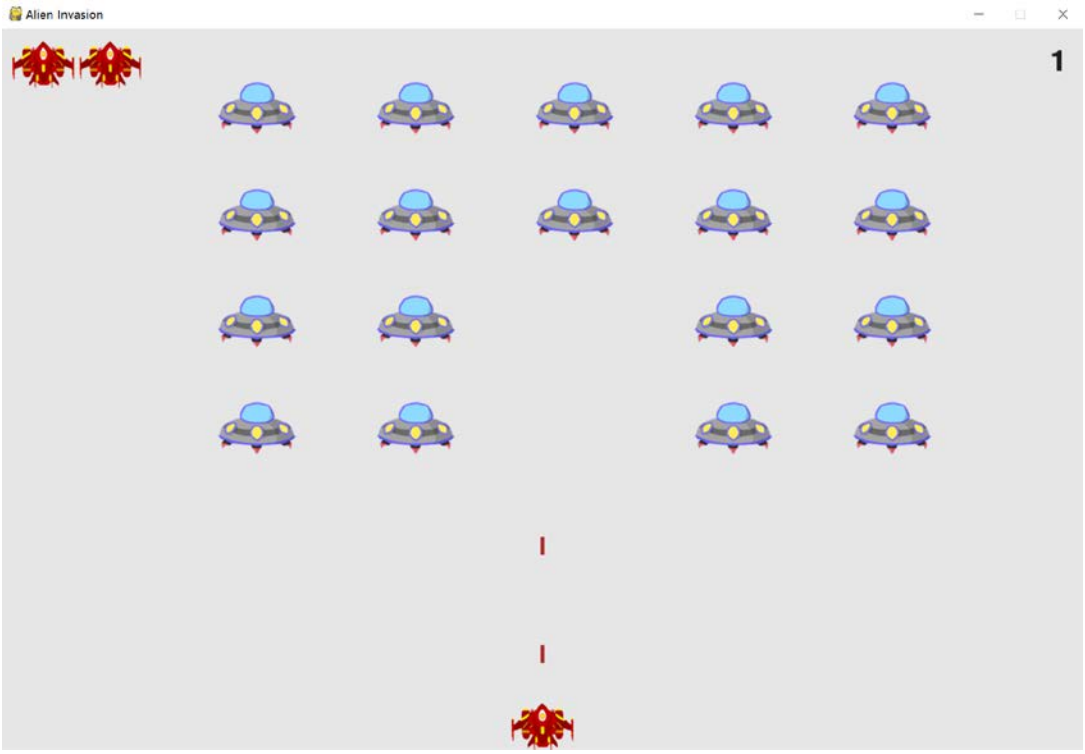
```
    #游戏的主循环
    while True:
        #……此处省略追踪部分
        gf.check_events(ai_settings, screen, stats, sb, play_button, ship, aliens, bullets)
        if stats.game_active: #如果游戏处于活动状态，更新下列内容
            gf.fire_bullet(ai_settings, screen, ship, bullets)
            ship.update()
            gf.update_bullets(ai_settings, screen, stats, sb, ship, aliens, bullets)
            gf.update_aliens(ai_settings, stats, screen, sb, ship, aliens, bullets)
        gf.update_screen(ai_settings, stats, screen, sb, ship, aliens, bullets, play_button)

run_game()
```

在主循环中，我们先调用 check_events 来检测并判定 stats.game_active 是否为 True。如果为 True，则各对象不断更新，调用各函数来达到这一效果。此外，update_screen 函数要放在 if 语句外，因为屏幕时刻都要被绘制出。

六、 结论

本项目中，通过我们对 opencv, pygame 的学习与实践，成功实现预期功能：通过 open 色彩追踪使飞机大战可以摆脱键盘来操控，为游戏增添了可玩性。在自学过程中，我们遇到了不少困难，但都通过讨论、搜索等方式解决。同时，图像识别效果好，准确度高，游戏过程流畅。游戏效果如下图所示。



本项目是小组成员第一次完成相关的程序设计项目，还有不少有待改进之处。由于知识储备和经验还不足，我们在项目中援引了不少代码。但本项目的锻炼让我们对程序设计有了更加深入的了解，产生了兴趣，希望以后在相关领域进一步学习。

七、 展望

1. 在游戏开始前，让用户输入可识别物体的颜色

目前只能利用红色的物体来控制飞船的移动来进行游戏，有时候缺少红色物体，或是穿着一件红色衣服，就可能无法进行游戏。可以通过构造一个颜色阈值的字典，在游戏开始前让用户选择颜色的种类，便可以避免这个问题的发生。

2. 游戏内容复杂化，设置新玩法

目前游戏内容较为单一，外星飞船移动方式简单。希望以后可以让敌机的移动方式变得复杂，或是在一定范围内随机移动，也可以让飞船上下移动，增加游戏的趣味性也增加操作上限。

3. 用手的移动来控制飞船移动

相比于拿着物体控制游戏，直接用手的移动来控制游戏更为简单简洁。但因为基于手的移动所需要的库复杂多样，需要提供大量的素材照片给计算机学习，在较短的时间内我们无法清晰明白其中的代码的运作方式，希望在后期能够完成通过对手的移动的检测来控制游戏。