《Python程序设计基础》程序设计作品说明书

题目： 外星人入侵游戏

学院： 21计科03

姓名： 蒋俊杰

学号： B20210302302

指导教师： 周景

起止日期：2023.11.10-2023.12.10

摘要

本项目旨在创建一个简单的外星人入侵游戏，使用Python编程语言和Pygame库。游戏的基本目标是让玩家控制飞船躲避外星人的攻击并尽可能击败更多外星人。本文详细介绍了游戏的设计和实现过程，包括飞船、外星人、子弹、游戏统计信息等关键组件的创建和交互。通过这个项目，我们不仅学习了游戏开发的基本概念，还深入了解了Python编程语言和Pygame库的使用。

关键词： 游戏开发、Python、Pygame、外星人入侵

关键词：

第1章 需求分析

1.1 项目背景

外星人入侵游戏是一种经典的街机游戏，玩家通过操控飞船躲避外星人的攻击，尽可能地击败更多的外星人。本项目旨在通过使用Python和Pygame库，实现一个简单而有趣的外星人入侵游戏，为用户提供轻松娱乐的游戏体验。

1.2 主要功能

本系统主要包括以下功能：

飞船控制： 玩家可以通过键盘控制飞船左右移动，躲避外星人的攻击。 射击功能： 玩家可以发射子弹来击败外星人，提高得分。 外星人生成： 外星人将在屏幕上随机生成，向玩家飞来。 得分统计： 游戏将记录玩家的得分，并显示最高得分。 游戏难度递增： 随着游戏的进行，外星人的移动速度将逐渐加快，提高游戏难度。

1.3 用户问题

本项目旨在提供一种娱乐方式，让用户在游戏中放松身心。通过挑战击败外星人，玩家可以获得成就感和娱乐体验。

1.4 系统环境

操作系统：支持运行Python的操作系统，如Windows、Linux、macOS等。 编程语言：Python 开发工具：Pygame库

1. 5 总结

通过分析，我们明确了项目的背景、主要功能以及用户问题。接下来的章节将详细介绍每个功能的实现过程，以及在开发中遇到的一些关键问题和解决方案。

第2章 分析与设计

2.1 系统架构

本系统采用经典的游戏开发架构，主要包括以下模块：

游戏引擎模块： 使用Pygame库作为游戏引擎，负责处理图形绘制、用户输入等底层操作。 游戏对象模块： 包括飞船、外星人、子弹等游戏中的各种对象，通过对象间的交互实现游戏逻辑。 游戏控制模块： 处理用户输入，控制游戏流程，如开始、暂停、重新开始等。 得分统计模块： 记录和显示玩家的得分、最高分以及游戏等级。 游戏设置模块： 包括游戏中的各项设置，如屏幕大小、飞船速度、子弹速度等。

2.2 数据结构

2.2.1 飞船（Ship）对象

属性：

图像（image）：飞船的图像。 外接矩形（rect）：飞船图像的外接矩形。 位置（center）：飞船的当前位置。 移动标志（moving_right、moving_left）：记录飞船是否向右或向左移动。 方法：

init(self, ai_settings, screen): 初始化飞船对象。 update(self): 根据移动标志更新飞船的位置。 blitme(self): 在指定位置绘制飞船。

2.2.2 外星人（Alien）对象

属性：

图像（image）：外星人的图像。 外接矩形（rect）：外星人图像的外接矩形。 位置（x, y）：外星人的当前位置。 速度因子（alien_speed_factor）：外星人的移动速度。 方法：

init(self, ai_settings, screen): 初始化外星人对象。 update(self): 向右移动外星人。 blitme(self): 在指定位置绘制外星人。 check_edges(self): 如果外星人位于屏幕边缘，返回True。

2.2.3 子弹（Bullet）对象

属性：

图像（image）：子弹的图像。 外接矩形（rect）：子弹图像的外接矩形。 位置（x, y）：子弹的当前位置。 速度因子（speed_factor）：子弹的移动速度。 方法：

init(self, ai_settings, screen, ship): 初始化子弹对象。 update(self): 向上移动子弹。 draw_bullet(self): 在屏幕上绘制子弹。

2.3 系统流程

整体游戏流程如下：

游戏初始化：设置游戏参数，创建游戏对象，加载图像资源等。 主循环开始：监听用户输入，更新游戏对象状态，检测碰撞等。 用户输入处理：响应键盘输入，控制飞船移动和射击。 更新游戏对象状态：更新飞船、外星人、子弹的位置和状态。 碰撞检测：检测子弹与外星人的碰撞，更新得分和最高分。 外星人生成：根据一定规则生成新的外星人。 游戏结束检测：检测飞船与外星人的碰撞，判断是否结束游戏。 刷新屏幕：根据最新的游戏状态，重新绘制屏幕。 主循环结束。

2.4 数据库设计

由于本项目为简单的单机游戏，不涉及数据库存储，得分统计和最高分采用本地文件进行保存和读取。

2.5 算法设计

2.5.1 子弹与外星人的碰撞检测

```
collisions = pygame.sprite.groupcollide(bullets, aliens, True, True)
if collisions:
    for aliens in collisions.values():
        stats.score += ai_settings.alien_points * len(aliens)
        sb.prep_score()
    check_high_score(stats, sb)
```

第3章 软件测试

3.1 单元测试

3.1.1 飞船（Ship）对象测试

测试用例 1：初始化飞船对象

```
def test_ship_init():
    ai_settings = Settings()
    screen = pygame.display.set_mode((ai_settings.screen_width,
ai_settings.screen_height))
    ship = Ship(ai_settings, screen)
    assert ship.rect.centerx == screen.get_rect().centerx
    assert ship.rect.bottom == screen.get_rect().bottom
    assert ship.moving_right == False
    assert ship.moving_left == False
```

测试用例 2：更新飞船位置

```
def test_ship_update():
    ai_settings = Settings()
    screen = pygame.display.set_mode((ai_settings.screen_width,
ai_settings.screen_height))
    ship = Ship(ai_settings, screen)
```

```
    initial_centerx = ship.rect.centerx
    ship.moving_right = True
    ship.update()
    assert ship.rect.centerx == initial_centerx + ai_settings.ship_speed_factor
```

3.1.2 外星人（Alien）对象测试

测试用例 3：初始化外星人对象

```
def test_alien_init():
    ai_settings = Settings()
    screen = pygame.display.set_mode((ai_settings.screen_width,
ai_settings.screen_height))
    alien = Alien(ai_settings, screen)
    assert alien.rect.x == alien.rect.width
    assert alien.rect.y == alien.rect.height
    assert alien.x == float(alien.rect.x)
    assert alien.check_edges() == False
```

测试用例 4：更新外星人位置

```
def test_alien_update():
    ai_settings = Settings()
    screen = pygame.display.set_mode((ai_settings.screen_width,
ai_settings.screen_height))
    alien = Alien(ai_settings, screen)
    initial_x = alien.x
    alien.update()
    assert alien.x == initial_x + (ai_settings.alien_speed_factor *
ai_settings.fleet_direction)
```

3.1.3 子弹（Bullet）对象测试

测试用例 5：初始化子弹对象

```
def test_bullet_init():
    ai_settings = Settings()
    screen = pygame.display.set_mode((ai_settings.screen_width,
ai_settings.screen_height))
    ship = Ship(ai_settings, screen)
    bullet = Bullet(ai_settings, screen, ship)
    assert bullet.rect.centerx == ship.rect.centerx
    assert bullet.rect.top == ship.rect.top
    assert bullet.y == float(bullet.rect.y)
```

测试用例 6：更新子弹位置

```
def test_bullet_update():
    ai_settings = Settings()
    screen = pygame.display.set_mode((ai_settings.screen_width,
ai_settings.screen_height))
    ship = Ship(ai_settings, screen)
    bullet = Bullet(ai_settings, screen, ship)
    initial_y = bullet.y
    bullet.update()
    assert bullet.y == initial_y - ai_settings.bullet_speed_factor
```

3.2 功能测试

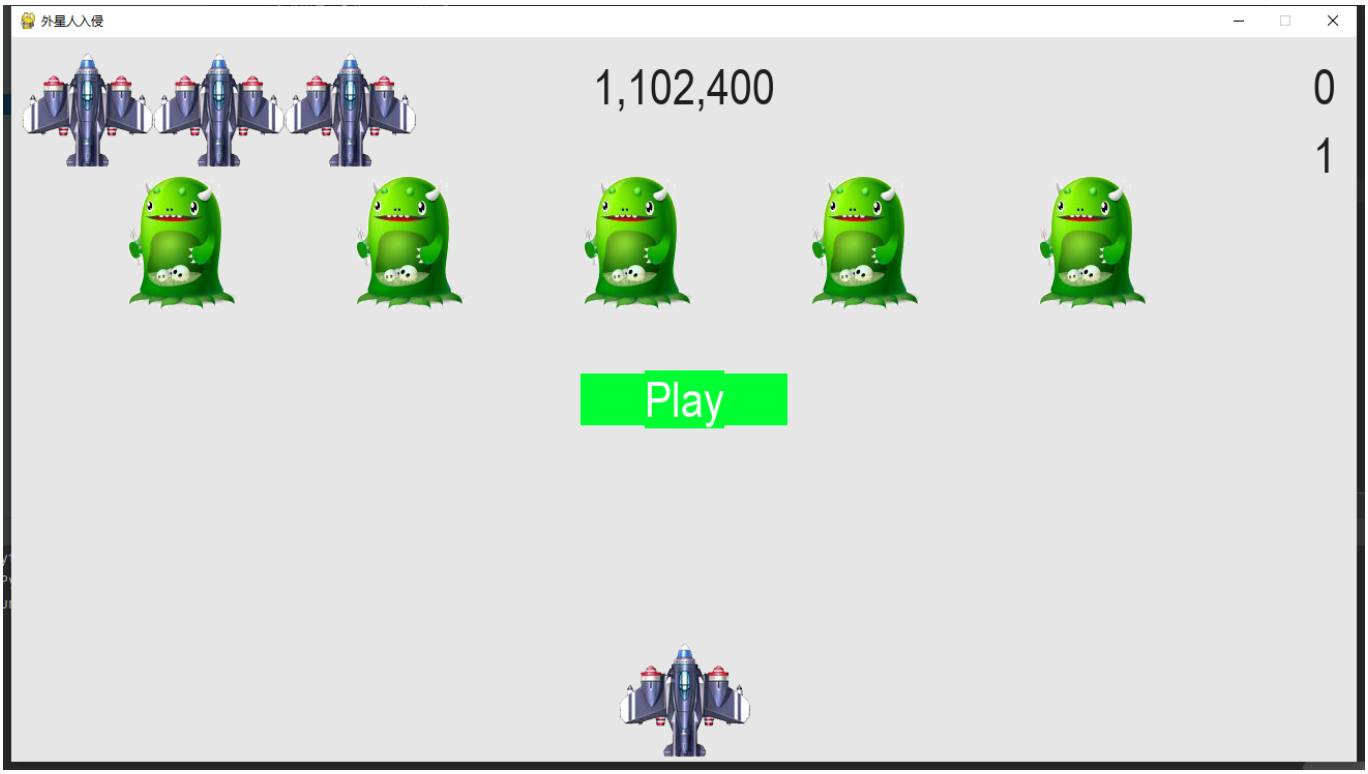
测试用例 7：检测外星人是否生成

```
def test_create_aliens():
    ai_settings = Settings()
    screen = pygame.display.set_mode((ai_settings.screen_width,
ai_settings.screen_height))
    ship = Ship(ai_settings, screen)
    aliens = Group()
    create_fleet(ai_settings, screen, ship, aliens)
    assert len(aliens) > 0
```

测试用例 8：检测得分统计是否正常

```
def test_score_count():
    ai_settings = Settings()
    screen = pygame.display.set_mode((ai_settings.screen_width,
ai_settings.screen_height))
    stats = GameStats(ai_settings)
    sb = Scoreboard(ai_settings, screen, stats)
    stats.score = 100
    sb.prep_score()
    assert sb.score_image.get_width() > 0
```

运行截图：



单元测试用例

#	测试目标	输入	预期结果	测试结果
1	初始化飞船对象		飞船位于屏幕底部中央	成功
2	更新飞船位置		飞船向右移动一定距离	成功
3	外星人对象测试		击杀外星人后，得分增加相应的分数，最高分记录	成功

结论

通过本次项目的设计与实现，成功实现了一个基于 Pygame 的外星人入侵游戏。在需求分析阶段明确定义了游戏的功能，包括飞船的移动、子弹的发射、外星人的生成和移动、得分统计、最高分记录等。在系统的设计过程中，采用了面向对象的方法，设计了飞船、子弹、外星人等类，并定义了相应的方法和属性。在测试阶段，编写了单元测试用例，验证了各个类的功能和逻辑的正确性。

然而，项目仍然存在一些不足之处。游戏目前比较简单，缺少一些复杂的游戏机制和关卡设计。在后续的改进中，可以考虑增加游戏关卡、优化游戏界面、增加更多的游戏元素等，以提升游戏的趣味性和可玩性。此外，可以考虑优化代码结构，提高代码的可维护性和可扩展性。

总体而言，本项目为初学者提供了一个基于 Pygame 的游戏开发实践，通过实际操作锻炼了编码能力和问题解决能力。在今后的学习和工作中，可以在此基础上进一步深入学习游戏开发、图形界面设计等相关领域，不断提升自己的技术水平。

参考文献