Alien Invasion final project documentation

Author: Ashley Ramos

Asheville Buncombe Technical Community College

CSC-121-OA1

Contents

[Overview: 2](#_Toc197291815)

[How to Use the Application: 2](#_Toc197291816)

[Resources Used: 3](#_Toc197291817)

[Video: 3](#_Toc197291818)

[Source Code: 4](#_Toc197291819)

[alien\_invasion.py 4](#_Toc197291820)

[settings.py 13](#_Toc197291821)

[game\_stats.py 15](#_Toc197291822)

[ship.py 16](#_Toc197291823)

[alien.py 17](#_Toc197291824)

[bullet.py 18](#_Toc197291825)

[button.py 20](#_Toc197291826)

[scoreboard.py 21](#_Toc197291827)

# Overview:

The *Alien Invasion* project is a 2D arcade-style shooting game developed using Python and Pygame. The player controls a spaceship that can move horizontally and fire bullets at descending alien ships. The goal is to eliminate all alien ships while avoiding collisions. This game incorporates principles from Object-Oriented Programming and event-driven game loops, developed by following Chapters 12 through 14 of the *Python Crash Course* textbook.

# How to Use the Application:

**Setup Instructions:**

1. Ensure Python 3.11+ is installed.
2. Install Pygame by running: *pip install pygame*
3. Run the main script: alien\_invasion.py

**Controls:**

* **Right Arrow:** Move ship right ->
* **Left Arrow:** Move ship left <-
* **Spacebar:** Fire bullets
* **P key:** Start the game
* **Q key:** Quit the game

**Game Flow:**

* Click “Play” to begin.
* Shoot all the aliens before they reach the bottom or collide with your ship.
* If you lose all ships, a “Game Over” message and a “Retry” button appear.
* Click “Retry” to restart the game.

A screenshot of a video game

AI-generated content may be incorrect.A screenshot of a video game

AI-generated content may be incorrect.

# Resources Used:

* *Python Crash Course* by Eric Matthes – Chapters 12, 13, 14
* Pygame Documentation: https://www.pygame.org/docs/
* Microsoft Visual Studio Code for development
* Open-source images used for space\_ship.png and alien\_space\_ship.png
* Online guides on managing game states and button interactions in Pygame

# Video:



# Source Code:

### alien\_invasion.py

from time import sleep

import sys

import pygame

from settings import Settings

from game\_stats import GameStats

from scoreboard import Scoreboard

from button import Button

from ship import Ship

from bullet import Bullet

from alien import Alien

class AlienInvasion:

    """Overall class to manage game assets and behavior."""

    def \_\_init\_\_(self):

        pygame.init()

        self.settings = Settings()

        self.screen = pygame.display.set\_mode(

            (self.settings.screen\_width, self.settings.screen\_height))

        pygame.display.set\_caption("Alien Invasion")

        # Create screen\_rect here

        self.screen\_rect = self.screen.get\_rect()

        # Load background

        self.background = pygame.image.load('space\_background.jpg')

        self.background = pygame.transform.scale(

            self.background,

            (self.settings.screen\_width, self.settings.screen\_height))

        self.stats = GameStats(self)

        self.sb = Scoreboard(self)

        self.ship = Ship(self)

        self.bullets = pygame.sprite.Group()

        self.aliens = pygame.sprite.Group()

        self.\_create\_fleet()

        self.play\_button = Button(self, "Play")

        # Game Over elements

        self.game\_over\_font = pygame.font.SysFont(None, 100)

        self.game\_over\_msg = "GAME OVER"

        self.game\_over\_image = None

        self.game\_over\_rect = None

        self.retry\_button = None

    def run\_game(self):

        while True:

            self.\_check\_events()

            if self.stats.game\_active:

                self.ship.update()

                self.\_update\_bullets()

                self.\_update\_aliens()

            self.\_update\_screen()

    def \_check\_events(self):

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                sys.exit()

            elif event.type == pygame.KEYDOWN:

                self.\_check\_keydown\_events(event)

            elif event.type == pygame.KEYUP:

                self.\_check\_keyup\_events(event)

            elif event.type == pygame.MOUSEBUTTONDOWN:

                mouse\_pos = pygame.mouse.get\_pos()

                if not self.stats.game\_active:

                    if self.stats.ships\_left == 0:  # Game over state

                        if self.retry\_button.rect.collidepoint(mouse\_pos):

                            self.\_start\_game()

                    else:  # Start screen

                        self.\_check\_play\_button(mouse\_pos)

    def \_show\_game\_over(self):

        """Display game over message and retry button."""

        self.game\_over\_image = self.game\_over\_font.render(

            self.game\_over\_msg, True, (255, 0, 0))  # Red text

        self.game\_over\_rect = self.game\_over\_image.get\_rect()

        self.game\_over\_rect.center = self.screen\_rect.center

        self.game\_over\_rect.y -= 100  # Move up from center

        # Create retry button below game over message

        if not self.retry\_button:

            self.retry\_button = Button(self, "Retry", y\_offset=50)

    def \_check\_play\_button(self, mouse\_pos):

        button\_clicked = self.play\_button.rect.collidepoint(mouse\_pos)

        if button\_clicked and not self.stats.game\_active:

            self.\_start\_game()

    def \_start\_game(self):

        self.settings.initialize\_dynamic\_settings()

        self.stats.reset\_stats()

        self.stats.game\_active = True

        self.sb.prep\_images()

        self.aliens.empty()

        self.bullets.empty()

        self.\_create\_fleet()

        self.ship.center\_ship()

        pygame.mouse.set\_visible(False)

    def \_check\_keydown\_events(self, event):

        if event.key == pygame.K\_RIGHT:

            self.ship.moving\_right = True

        elif event.key == pygame.K\_LEFT:

            self.ship.moving\_left = True

        elif event.key == pygame.K\_q:

            sys.exit()

        elif event.key == pygame.K\_SPACE:

            self.\_fire\_bullet()

        elif event.key == pygame.K\_p and not self.stats.game\_active:

            self.\_start\_game()

    def \_check\_keyup\_events(self, event):

        if event.key == pygame.K\_RIGHT:

            self.ship.moving\_right = False

        elif event.key == pygame.K\_LEFT:

            self.ship.moving\_left = False

    def \_fire\_bullet(self):

        if len(self.bullets) < self.settings.bullets\_allowed:

            new\_bullet = Bullet(self)

            self.bullets.add(new\_bullet)

    def \_update\_bullets(self):

        self.bullets.update()

        for bullet in self.bullets.copy():

            if bullet.rect.bottom <= 0:

                self.bullets.remove(bullet)

        self.\_check\_bullet\_alien\_collisions()

    def \_check\_bullet\_alien\_collisions(self):

        collisions = pygame.sprite.groupcollide(

            self.bullets, self.aliens, True, True)

        if collisions:

            for aliens in collisions.values():

                self.stats.score += self.settings.alien\_points \* len(aliens)

            self.sb.prep\_score()

            self.sb.check\_high\_score()

        if not self.aliens:

            self.\_start\_new\_level()

    def \_start\_new\_level(self):

        self.bullets.empty()

        self.\_create\_fleet()

        self.settings.increase\_speed()

        self.stats.level += 1

        self.sb.prep\_level()

    def \_update\_aliens(self):

        self.\_check\_fleet\_edges()

        self.aliens.update()

        if pygame.sprite.spritecollideany(self.ship, self.aliens):

            self.\_ship\_hit()

        self.\_check\_aliens\_bottom()

    def \_check\_fleet\_edges(self):

        for alien in self.aliens.sprites():

            if alien.check\_edges():

                self.\_change\_fleet\_direction()

                break

    def \_change\_fleet\_direction(self):

        for alien in self.aliens.sprites():

            alien.rect.y += self.settings.fleet\_drop\_speed

        self.settings.fleet\_direction \*= -1

    def \_ship\_hit(self):

        if self.stats.ships\_left > 0:

            self.stats.ships\_left -= 1

            self.sb.prep\_ships()

            self.aliens.empty()

            self.bullets.empty()

            self.\_create\_fleet()

            self.ship.center\_ship()

            sleep(0.5)

        else:

            self.stats.game\_active = False

            pygame.mouse.set\_visible(True)

            self.\_show\_game\_over()

    def \_check\_aliens\_bottom(self):

        screen\_rect = self.screen.get\_rect()

        for alien in self.aliens.sprites():

            if alien.rect.bottom >= screen\_rect.bottom:

                self.\_ship\_hit()

                break

    def \_create\_fleet(self):

        alien = Alien(self)

        alien\_width, alien\_height = alien.rect.size

        available\_space\_x = self.settings.screen\_width - (2 \* alien\_width)

        number\_aliens\_x = available\_space\_x // (2 \* alien\_width)

        ship\_height = self.ship.rect.height

        available\_space\_y = (self.settings.screen\_height -

                            (3 \* alien\_height) - ship\_height)

        number\_rows = available\_space\_y // (2 \* alien\_height)

        for row\_number in range(number\_rows):

            for alien\_number in range(number\_aliens\_x):

                self.\_create\_alien(alien\_number, row\_number)

    def \_create\_alien(self, alien\_number, row\_number):

        alien = Alien(self)

        alien\_width, alien\_height = alien.rect.size

        alien.x = alien\_width + 2 \* alien\_width \* alien\_number

        alien.rect.x = alien.x

        alien.rect.y = alien.rect.height + 2 \* alien.rect.height \* row\_number

        self.aliens.add(alien)

    def \_update\_screen(self):

        self.screen.blit(self.background, (0, 0))

        self.ship.blitme()

        for bullet in self.bullets.sprites():

            bullet.draw\_bullet()

        self.aliens.draw(self.screen)

        self.sb.show\_score()

        if not self.stats.game\_active:

            if self.stats.ships\_left == 0:  # Game Over state

                self.screen.blit(self.game\_over\_image, self.game\_over\_rect)

                self.retry\_button.draw\_button()

            else:  # Start screen

                self.play\_button.draw\_button()

        pygame.display.flip()

if \_\_name\_\_ == '\_\_main\_\_':

    ai = AlienInvasion()

    ai.run\_game()

### settings.py

class Settings:

    """A class to store all settings for Alien Invasion."""

    def \_\_init\_\_(self):

        """Initialize the game's static settings."""

        # Screen settings

        self.screen\_width = 1200

        self.screen\_height = 800

        self.bg\_color = (230, 230, 230)

        # Ship settings

        self.ship\_limit = 3

        # Bullet settings

        self.bullet\_width = 3

        self.bullet\_height = 15

        self.bullet\_color = (60, 60, 60)

        self.bullets\_allowed = 3

        # Alien settings

        self.fleet\_drop\_speed = 10

        # How quickly the game speeds up

        self.speedup\_scale = 1.1

        # How quickly the alien point values increase

        self.score\_scale = 1.5

        self.initialize\_dynamic\_settings()

    def initialize\_dynamic\_settings(self):

        """Initialize settings that change throughout the game."""

        self.ship\_speed = 1.5

        self.bullet\_speed = 3.0

        self.alien\_speed = 1.0

        # fleet\_direction of 1 represents right; -1 represents left.

        self.fleet\_direction = 1

        # Scoring

        self.alien\_points = 50

    def increase\_speed(self):

        """Increase speed settings and alien point values."""

        self.ship\_speed \*= self.speedup\_scale

        self.bullet\_speed \*= self.speedup\_scale

        self.alien\_speed \*= self.speedup\_scale

        self.alien\_points = int(self.alien\_points \* self.score\_scale)

### game\_stats.py

class GameStats:

    """Track statistics for Alien Invasion."""

    def \_\_init\_\_(self, ai\_game):

        """Initialize statistics."""

        self.settings = ai\_game.settings

        self.reset\_stats()

        # High score should never be reset

        self.high\_score = 0

    def reset\_stats(self):

        """Initialize statistics that can change during the game."""

        self.ships\_left = self.settings.ship\_limit

        self.score = 0

        self.level = 1

        self.game\_active = False  # Start inactive after reset

### ship.py

import pygame

from pygame.sprite import Sprite

class Ship(Sprite):

    def \_\_init\_\_(self, ai\_game):

        super().\_\_init\_\_()

        self.screen = ai\_game.screen

        self.settings = ai\_game.settings

        self.screen\_rect = ai\_game.screen.get\_rect()

        # Load ship image

        try:

            self.image = pygame.image.load('space\_ship.png')

            self.image = pygame.transform.scale(self.image, (64, 64))

        except:

            self.image = pygame.Surface((50, 30))

            self.image.fill((0, 255, 0))

        self.rect = self.image.get\_rect()

        self.rect.midbottom = self.screen\_rect.midbottom

        self.x = float(self.rect.x)

        self.moving\_right = False

        self.moving\_left = False

    def update(self):

        if self.moving\_right and self.rect.right < self.screen\_rect.right:

            self.x += self.settings.ship\_speed

        if self.moving\_left and self.rect.left > 0:

            self.x -= self.settings.ship\_speed

        self.rect.x = self.x

    def blitme(self):

        self.screen.blit(self.image, self.rect)

    def center\_ship(self):

        self.rect.midbottom = self.screen\_rect.midbottom

        self.x = float(self.rect.x)

### alien.py

import pygame

from pygame.sprite import Sprite

class Alien(Sprite):

    def \_\_init\_\_(self, ai\_game):

        super().\_\_init\_\_()

        self.screen = ai\_game.screen

        self.settings = ai\_game.settings

        # Load alien image

        try:

            self.image = pygame.image.load('alien\_space\_ship.png')

            self.image = pygame.transform.scale(self.image, (60, 40))

        except:

            self.image = pygame.Surface((40, 40))

            self.image.fill((255, 0, 0))

        self.rect = self.image.get\_rect()

        self.rect.x = self.rect.width

        self.rect.y = self.rect.height

        self.x = float(self.rect.x)

    def check\_edges(self):

        screen\_rect = self.screen.get\_rect()

        return (self.rect.right >= screen\_rect.right) or (self.rect.left <= 0)

    def update(self):

        self.x += (self.settings.alien\_speed \* self.settings.fleet\_direction)

        self.rect.x = self.x

### bullet.py

import pygame

from pygame.sprite import Sprite

class Bullet(Sprite):

    """A class to manage bullets fired from the ship"""

    def \_\_init\_\_(self, ai\_game):

        """Create a bullet object at the ship's current position."""

        super().\_\_init\_\_()

        self.screen = ai\_game.screen

        self.settings = ai\_game.settings

        self.color = self.settings.bullet\_color

        # Create a bullet rect at (0, 0) and then set correct position

        self.rect = pygame.Rect(0, 0, self.settings.bullet\_width,

            self.settings.bullet\_height)

        self.rect.midtop = ai\_game.ship.rect.midtop

        # Store the bullet's position as a decimal value

        self.y = float(self.rect.y)

    def update(self):

        """Move the bullet up the screen."""

        # Update the decimal position of the bullet

        self.y -= self.settings.bullet\_speed

        # Update the rect position

        self.rect.y = self.y

    def draw\_bullet(self):

        """Draw the bullet to the screen."""

        pygame.draw.rect(self.screen, self.color, self.rect)

### button.py

import pygame.font

class Button:

    def \_\_init\_\_(self, ai\_game, msg, y\_offset=0):

        """Initialize button attributes with optional vertical offset."""

        self.screen = ai\_game.screen

        self.screen\_rect = self.screen.get\_rect()

        # Set dimensions and properties

        self.width, self.height = 200, 50

        self.button\_color = (0, 255, 0)  # Green

        self.text\_color = (255, 255, 255)

        self.font = pygame.font.SysFont(None, 48)

        # Build button's rect and center it with offset

        self.rect = pygame.Rect(0, 0, self.width, self.height)

        self.rect.center = self.screen\_rect.center

        self.rect.y += y\_offset  # Apply vertical offset

        # Prep button message

        self.\_prep\_msg(msg)

    def \_prep\_msg(self, msg):

        """Turn msg into a rendered image and center text on button."""

        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):

        """Draw blank button and then draw message."""

        self.screen.fill(self.button\_color, self.rect)

        self.screen.blit(self.msg\_image, self.msg\_image\_rect)

### scoreboard.py

import pygame.font

from pygame.sprite import Group

from ship import Ship

class Scoreboard:

    """A class to report scoring information."""

    def \_\_init\_\_(self, ai\_game):

        self.ai\_game = ai\_game

        self.screen = ai\_game.screen

        self.screen\_rect = self.screen.get\_rect()

        self.settings = ai\_game.settings

        self.stats = ai\_game.stats

        self.text\_color = (255, 255, 255)  # White text

        self.font = pygame.font.SysFont(None, 48)

        self.prep\_images()

    def prep\_images(self):

        self.prep\_score()

        self.prep\_high\_score()

        self.prep\_level()

        self.prep\_ships()

    def prep\_score(self):

        rounded\_score = round(self.stats.score, -1)

        score\_str = f"Score: {rounded\_score:,}"

        self.score\_image = self.font.render(score\_str, True, self.text\_color)

        self.score\_rect = self.score\_image.get\_rect()

        self.score\_rect.right = self.screen\_rect.right - 20

        self.score\_rect.top = 20

    def prep\_high\_score(self):

        high\_score = round(self.stats.high\_score, -1)

        high\_score\_str = f"High Score: {high\_score:,}"

        self.high\_score\_image = self.font.render(high\_score\_str, True, self.text\_color)

        self.high\_score\_rect = self.high\_score\_image.get\_rect()

        self.high\_score\_rect.centerx = self.screen\_rect.centerx

        self.high\_score\_rect.top = self.score\_rect.top

    def prep\_level(self):

        level\_str = f"Level: {self.stats.level}"

        self.level\_image = self.font.render(level\_str, True, self.text\_color)

        self.level\_rect = self.level\_image.get\_rect()

        self.level\_rect.right = self.score\_rect.right

        self.level\_rect.top = self.score\_rect.bottom + 10

    def prep\_ships(self):

        self.ships = Group()

        for ship\_number in range(self.stats.ships\_left):

            ship = Ship(self.ai\_game)

            ship.rect.x = 10 + ship\_number \* ship.rect.width

            ship.rect.y = 10

            self.ships.add(ship)

    def check\_high\_score(self):

        if self.stats.score > self.stats.high\_score:

            self.stats.high\_score = self.stats.score

            self.prep\_high\_score()

    def show\_score(self):

        """Draw scores, level, and ships to the screen."""

        if self.stats.game\_active or self.stats.ships\_left > 0:

            self.screen.blit(self.score\_image, self.score\_rect)

            self.screen.blit(self.high\_score\_image, self.high\_score\_rect)

            self.screen.blit(self.level\_image, self.level\_rect)

            self.ships.draw(self.screen)