**Alien Invasion Game**

**Introduction**

Alien Invasion is a 2D arcade-style shooting game where the player controls a spaceship at the bottom of the screen. The goal is to shoot down waves of descending aliens before they reach the bottom or collide with your ship.

**Game Features:**

* A fleet of aliens that increases in speed with each level.
* Bullet firing with a limited number on screen at once.
* Score tracking, level advancement, and high score display.
* Game Over when the player loses all three lives.

**Requirements**

* [Python 3.x](https://www.python.org/downloads/)
* [Pygame Library](https://www.pygame.org/news)

**Controls**

|  |  |
| --- | --- |
| **Key** | **Action** |
| Right Arrow | Move ship right |
| Left Arrow | Move ship left |
| Spacebar | Shoot bullet |

**How to Run**

1. Clone or download the project folder.
2. Open a terminal in the project directory.
3. Run the game with the following command: python alien\_invasion.py

**Screenshots**

**Game Start Screen**

**A screenshot of a video game

AI-generated content may be incorrect.**

**GameplayA screenshot of a video game

AI-generated content may be incorrect.**

**Game Over**

A screenshot of a video game

AI-generated content may be incorrect.

**Resources Used**

* **Python Crash Course** by Eric Matthes – Primary reference for game structure.
* [Pygame Documentation](https://www.pygame.org/docs/)
* Assets: Ship image and alien sprites are custom/placeholder images.

**Code Printout**

**alien\_invasion.py – Main game loop and structure**

import sys

import pygame

from time import sleep

from settings import Settings

from ship import Ship

from bullet import Bullet

from alien import Alien

from game\_stats import GameStats

from scoreboard import Scoreboard

from button import Button

class AlienInvasion:

    """Class to manage game resources and behavior."""

    def \_\_init\_\_(self):

        """Initialize the game and create game resources."""

        pygame.init()

        self.settings = Settings()

        self.screen = pygame.display.set\_mode((0, 0), pygame.FULLSCREEN)

        self.settings.screen\_width = self.screen.get\_rect().width

        self.settings.screen\_height = self.screen.get\_rect().height

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

        # Create instances for statistics and scoreboard

        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()

        # Create the Play button

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

    def run\_game(self):

        """Start the main game loop."""

        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):

        """Respond to keyboard and mouse events."""

        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()

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

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

        """Respond to keypresses."""

        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()

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

        """Respond to key releases."""

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

            self.ship.moving\_right = False

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

            self.ship.moving\_left = False

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

        """Start a new game when the Play button is clicked."""

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

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

            # Reset game settings and stats

            self.stats.reset\_stats()

            self.settings.initialize\_dynamic\_settings()

            self.stats.game\_active = True

            self.sb.prep\_score()

            self.sb.prep\_level()

            # Clear aliens and bullets

            self.aliens.empty()

            self.bullets.empty()

            # Create a new fleet and center the ship

            self.\_create\_fleet()

            self.ship.center\_ship()

            # Hide the mouse cursor

            pygame.mouse.set\_visible(False)

    def \_fire\_bullet(self):

        """Create a new bullet and add it to the bullets group."""

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

            new\_bullet = Bullet(self)

            self.bullets.add(new\_bullet)

    def \_update\_bullets(self):

        """Update bullet positions and remove old bullets."""

        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):

        """Handle collisions between bullets and aliens."""

        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()

            self.sb.prep\_level()

            self.sb.prep\_ships()

        if not self.aliens:

            # Destroy existing bullets and create new fleet

            self.bullets.empty()

            self.\_create\_fleet()

            self.settings.increase\_speed()

            # Increase level

            self.stats.level += 1

            self.sb.prep\_level()

    def \_create\_fleet(self):

        """Create the alien fleet."""

        alien = Alien(self)

        self.aliens.add(alien)

        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)

        # Determine number of rows that fit on the screen

        ship\_height = self.ship.rect.height

        available\_space\_y = self.settings.screen\_height - (3 \* alien\_height) - ship\_height

        number\_rows = available\_space\_y // (3 \* alien\_height)

        # Create the fleet

        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):

        """Create an alien and place it in the row."""

        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\_height \* row\_number

        self.aliens.add(alien)

    def \_check\_fleet\_edges(self):

        """Respond if any aliens have reached the edge of the screen."""

        for alien in self.aliens.sprites():

            if alien.check\_edges():

                self.\_change\_fleet\_direction()

                break

    def \_change\_fleet\_direction(self):

        """Drop the fleet and change its direction."""

        for alien in self.aliens.sprites():

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

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

    def \_update\_aliens(self):

        """Update the positions of all aliens in the fleet."""

        self.\_check\_fleet\_edges()

        self.aliens.update()

        # Look for alien-ship collisions

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

            self.\_ship\_hit()

        # Check if aliens reached the bottom of the screen

        self.\_check\_aliens\_bottom()

    def \_ship\_hit(self):

        """Respond to the ship being hit by an alien."""

        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(1)

        else:

            self.stats.game\_active = False

            pygame.mouse.set\_visible(True)

    def \_check\_aliens\_bottom(self):

        """Check if any aliens have reached the bottom of the screen."""

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

        for alien in self.aliens.sprites():

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

                # Same as ship collision

                self.\_ship\_hit()

                break

    def \_update\_screen(self):

        """Update the screen and flip to the new frame."""

        self.screen.fill(self.settings.bg\_color)

        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:

            self.play\_button.draw\_button()

        pygame.display.flip()

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

    # Create a game instance and run the game

    ai = AlienInvasion()

    ai.run\_game()

**settings.py – Game configuration**

class Settings:

"""A class to store all settings for the game."""

def \_\_init\_\_(self):

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

# Screen settings

self.screen\_width = 1280

self.screen\_height = 720

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

# Ship settings

self.ship\_speed = 5

self.ship\_limit = 3

# Bullet settings

self.bullet\_speed = 4

self.bullet\_width = 5

self.bullet\_height = 20

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

self.bullets\_allowed = 3

# Alien settings

self.alien\_speed = 2

self.fleet\_drop\_speed = 10

# fleet\_direction = 1 means right; -1 means left

self.fleet\_direction = 1

# Game speed-up scale

self.speedup\_scale = 1.1

# Alien score growth scale

self.score\_scale = 1.5

self.initialize\_dynamic\_settings()

def initialize\_dynamic\_settings(self):

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

self.ship\_speed\_factor = 1.5

self.bullet\_speed\_factor = 3.0

self.alien\_speed\_factor = 1

# Scoring

self.alien\_points = 50

def increase\_speed(self):

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

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

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

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

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

**game\_stats.py – Score and level tracking**

class GameStats():

"""Track statistics for the game Alien Invasion."""

def \_\_init\_\_(self, ai\_game):

"""Initialize statistics."""

self.settings = ai\_game.settings

self.reset\_stats()

self.high\_score = 0

# The game starts in an inactive state

self.game\_active = False

def reset\_stats(self):

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

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

self.score = 0

self.level = 1

**scoreboard.py – Display elements**

import pygame.font

from pygame.sprite import Group

from ship import Ship

class Scoreboard:

"""Class for displaying game information."""

def \_\_init\_\_(self, ai\_game):

"""Initialize scoring attributes."""

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

# Font settings for displaying the score

self.text\_color = (30, 30, 30)

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

# Prepare the initial score images

self.prep\_score()

self.prep\_high\_score()

self.prep\_level()

self.prep\_ships()

def prep\_score(self):

"""Turn the current score into a rendered image."""

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

score\_str = "{:,}".format(rounded\_score)

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

# Display the score at the top right of the screen

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):

"""Turn the high score into a rendered image."""

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

high\_score\_str = "{:,}".format(high\_score)

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

# Center the high score at the top of the screen

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):

"""Turn the level into a rendered image."""

level\_str = str(self.stats.level)

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

# Position the level below the score

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):

"""Show how many ships are left."""

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 show\_score(self):

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

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)

def check\_high\_score(self):

"""Check to see if there's a new high score."""

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

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

self.prep\_high\_score()

**button.py – Start button**

import pygame.font

class Button:

def \_\_init\_\_(self, ai\_game, msg):

"""Initialize button attributes."""

self.screen = ai\_game.screen

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

# Set the dimensions and properties of the button

self.width, self.height = 200, 50

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

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

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

# Build the button's rect object and center it on the screen

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

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

# The button message is prepared only once

self.\_prep\_msg(msg)

def \_prep\_msg(self, msg):

"""Turn msg into a rendered image and center text on the 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 the button and then draw the message

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

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

**bullet.py – Bullet mechanics**

import pygame

from pygame.sprite import Sprite

class Bullet(Sprite):

"""Class to manage bullets fired from the ship."""

def \_\_init\_\_(self, ai\_game):

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

super(Bullet, self).\_\_init\_\_()

self.screen = ai\_game.screen

self.settings = ai\_game.settings

self.color = self.settings.bullet\_color

# Create a bullet at position (0, 0) and set the 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 float

self.y = float(self.rect.y)

def update(self):

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

# Update the float 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 on the screen."""

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

**ship.py – Player ship control**

import pygame

from pygame.sprite import Sprite

class Ship(Sprite):

"""A class to manage the ship."""

def \_\_init\_\_(self, ai\_game):

"""Initialize the ship and set its starting position."""

super().\_\_init\_\_()

self.screen = ai\_game.screen

self.settings = ai\_game.settings

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

# Load the ship image and get its rect.

self.image = pygame.image.load('images/ship.bmp')

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

# Start each new ship at the bottom center of the screen.

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

# Store a decimal value for the ship's horizontal position.

self.x = float(self.rect.x)

# Movement flags

self.moving\_right = False

self.moving\_left = False

def update(self):

"""Update the ship’s position based on movement flags."""

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

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

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

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

# Update rect object from self.x

self.rect.x = self.x

def blitme(self):

"""Draw the ship at its current location."""

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

def center\_ship(self):

"""Center the ship on the screen."""

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

self.x = float(self.rect.x)

**alien.py – Alien behavior**

import pygame

from pygame.sprite import Sprite

class Alien(Sprite):

"""A class to represent a single alien."""

def \_\_init\_\_(self, ai\_game):

"""Initialize the alien and set its starting position."""

super().\_\_init\_\_()

self.screen = ai\_game.screen

self.settings = ai\_game.settings

# Load the alien image and set its rect attribute.

self.image = pygame.image.load('images/alien.bmp')

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

# Start each new alien near the top-left of the screen.

self.rect.x = self.rect.width

self.rect.y = self.rect.height

# Store the alien's exact horizontal position.

self.x = float(self.rect.x)

def update(self):

"""Move the alien right or left."""

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

self.rect.x = self.x

def check\_edges(self):

"""Return True if alien is at edge of screen."""

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

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