How To Use: Alien Invasion

Tools and resources used: Python Crash Course, Third Edition By Eric Matthes; ChatGPT Version 4.0.

Start up:

To start the game, run the Alien_Invasion.py file with a compressor.

Explanation:

For the game to work we require all of the following elements: Alien, Ship, Settings, Stats, and Bullet classes and assets.

Here is the code for the Alien class:

```
import pygame

from pygame.sprite import Sprite


class Alien(Sprite):
   """A class to represent a single alien in the fleet."""

   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 check_edges(self):
       """Returns true if the alien is at the edge of the screen."""
       screen_rect = self.screen.get_rect()
       return(self.rect.right >= screen_rect.right) or (self.rect.left <= 0)

   def update(self):
       """Move the alien to right or left"""
       self.x += self.settings.alien_speed * self.settings.fleet_direction
       self.rect.x = self.x

   def blitme(self):
       """Draw the alien at its current location."""
       self.screen.blit(self.image,self.rect)
```

Here is the code for the Ship class:

```
import pygame

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

   def __init__(self, ai_game):
       """Initialize the ship and set its starting position."""
       self.screen = ai_game.screen
       self.screen_rect = ai_game.screen.get_rect()
       self.settings = ai_game.settings

       #Load the ship image and get its rect.
       self.image = pygame.image.load('/Users/joseochoa/Desktop/Python Coding
Class/Projects/Alien Invasion/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 float for the ship's exact horizontal position.
       self.x = float(self.rect.x)

       # Movement flag; start with a ship that's not moving.
       self.moving_right = False
       self.moving_left = False
```

```
def center_ship(self):
    """Center the ship on the screen."""
    self.rect.midbottom = self.screen_rect.midbottom
    self.x = float(self.rect.x)

def update(self):
    """Update the ship's position based on the movement flag."""
    # Update the ship's x value, not the rect.
    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
        # 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)
```

Here is the code for the Settings class:

```
class Settings:
   """A class to store all settings for Alien Invasion."""

   def __init__(self):
       """Initialize the game's settings."""
   # Screen settings
       self.screen_width = 1200
       self.screen_height = 800
       self.bg_color = (230,230,230)

       # Ship settings
       self.ship_speed = 1.5

       #bullet settings
       self.bullet_speed = 3.0
       self.bullet_width = 10
       self.bullet_height = 15
       self.bullet_color = (60,60,60)
```

```python
        self.bullets_allowed = 20

        # Alien Settings
        self.alien_speed = 1
        self.fleet_drop_speed = 5
        # fleet_direction of 1 represents right; -1 represents left.
        self.fleet_direction = 1
        self.ship_limit = 3
```

Here is the code for the Stats class:

```python
class Gamestats:
    """Track statistics for Alien Invasion."""

    def __init__(self, ai_game):
        """Initialize statistics."""
        self.settings = ai_game.settings
        self.reset_stats()

    def reset_stats(self):
        """Initialize statistics that can change during the game."""
        self.ships_left = self.settings.ship_limit
```

Here is the code for the Bullets class:

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

    def update(self):
        """Move the bullet up the screen."""
        # Update the exact 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)
```

We combine all of these In the main project file using logic and methods provided by the pygame directory.

All together the code will look like this:

```
import sys
from time import sleep

from settings import Settings

from alien import Alien
from ship import Ship
from game_stats import Gamestats
from bullet import Bullet

import pygame

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

   def __init__(self):
       """Initialize the game, and create game resources."""
```

```
        pygame.init()
        self.clock = pygame.time.Clock()
        self.settings = Settings()

        # Dynamically get the display's width and height
        display_info = pygame.display.Info()
        self.settings.screen_width = display_info.current_w
        self.settings.screen_height = display_info.current_h

        # Set the game to fullscreen mode with the display's dimensions
        self.screen = pygame.display.set_mode(
            (self.settings.screen_width, self.settings.screen_height),
pygame.FULLSCREEN
        )
        pygame.display.set_caption("Alien Invasion")

        # Create an instance of store game statistics.
        self.stats = Gamestats(self)

        # Create the ship
        self.ship = Ship(self)
        self.bullets = pygame.sprite.Group()
        self.aliens = pygame.sprite.Group()

        # Creates alien fleet
        self._create_fleet()

        # Set the background color
        self.bg_color = (230, 230, 230)

        # Start Alien Invasion in an active state.
        self.game_active = True

    def run_game(self):
        """Start the main loop for the game."""
        while True:
            #functions that are called to check keypresses and releases, update screen
elements and ship position on the screen.
            self._check_events()
            self._update_screen()

            if self.game_active:
```

```python
                self.ship.update()
                self._update_bullets()
                self._update_aliens()
            else:
                self._show_game_over()
            # Refreshes the Frame
            self.clock.tick(60)

    # Method that checks keypresses.
    def _check_events(self):
        """Respond to keypresses 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)

    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 _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 the position of bullets and get rid of old bullets."""
    # update bullet positions:
    self.bullets.update()
    # Get rid of bullets that have disappeared.
    for bullet in self.bullets.copy():
        if bullet.rect.bottom <= 0:
            self.bullets.remove(bullet)
            print(len(self.bullets))
    self._check_bullet_alien_collisions()

def _check_bullet_alien_collisions(self):
    """Respond to bullet-alien collisions."""
    # Remove any bullets and aliens that have collided.
    collisions = pygame.sprite.groupcollide(
            self.bullets, self.aliens, True, True)

    if collisions:
        print(f"Aliens remaining: {len(self.aliens)}")

    if len(self.aliens) == 0:
        # Destroy existing bullets and create a new fleet.
        print("All aliens destroyed. Respawning fleet...")
        self.bullets.empty()
        self._create_fleet()

def _ship_hit(self):
    """Respond to the ship being hit by an alien."""
    if self.stats.ships_left > 0:
        # Decrement ships_left.
        self.stats.ships_left -= 1

        # Get rid of any remaining bullets and aliens.
        self.bullets.empty()
        self.aliens.empty()

        # Create a new fleet and center the ship.
        self._create_fleet()
        self.ship.center_ship()

        # Pause.
```

```
            sleep(0.5)
        else:
            self.game_active = False



        # Check for bullets that have hit aliens.
        # If so, get rid of the bullet and the alien.
        collisions = pygame.sprite.groupcollide( self.bullets, self.aliens, True, True)

    def _update_aliens(self):
        """ Check if fleet is at the edge of the screen then update the positions of
the 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()
        # Look for aliens hitting the bottom of the screen.
        self._check_aliens_bottom()

    def _check_aliens_bottom(self):
        """Check if any aliens have reached the bottom of the screen."""
        for alien in self.aliens.sprites():
            if alien.rect.bottom >= self.settings.screen_height:
                # Treat this the same as if the ship got hit.
                self._ship_hit()
                break

    def _create_fleet(self):
        """Create the fleet of aliens."""
        # Create an alien and keep adding aliens until there's no room left.
        # Spacing between aliens is one alien width and one alien height.
        alien = Alien(self)
        alien_width, alien_height = alien.rect.size

        current_x, current_y = alien_width, alien_height
        while current_y < (self.settings.screen_height - 3 * alien_height):
            while current_x < (self.settings.screen_width - 2 * alien_width):
                self._create_alien(current_x, current_y)
                current_x += 2 * alien_width
        # Finished a row; reset x value, and increment y value.
        current_x = alien_width
        current_y += 2 * alien_height
    print("New fleet created.")

def _create_alien(self, x_position, y_position):
    """Create an alien and place it in the fleet."""
    new_alien = Alien(self)
    new_alien.x = x_position
    new_alien.rect.x = x_position
    new_alien.rect.y = y_position
    self.aliens.add(new_alien)

def _check_fleet_edges(self):
    """Respond appropriately if any aliens have reached an edge."""
    for alien in self.aliens.sprites():
        if alien.check_edges():
            self._change_fleet_direction()
            break

def _change_fleet_direction(self):
    """Drop the entire fleet and change the fleet's direction."""
    for alien in self.aliens.sprites():
        alien.rect.y += self.settings.fleet_drop_speed
    self.settings.fleet_direction *= -1

#Method that updates screen images
def _update_screen(self):
     """Update images on the screen, and flip to the new screen."""
     self.screen.fill(self.settings.bg_color)
     for bullet in self.bullets.sprites():
         bullet.draw_bullet()
     self.ship.blitme()
     self.aliens.draw(self.screen)

     # Make the most recently drawn screen visible.
     pygame.display.flip()

def _show_game_over(self):
    """Display the Game Over message."""
    font = pygame.font.SysFont(None, 74)  # Choose a font and size
    game_over_text = font.render("GAME OVER", True, (255, 0, 0))  # Red text
```

```
       text_rect = game_over_text.get_rect(center=(self.settings.screen_width // 2,
self.settings.screen_height // 2))
       self.screen.blit(game_over_text, text_rect)
       pygame.display.flip()  # Update the screen
       sleep(2)  # Pause for 2 seconds

if __name__ == '__main__':
   # Make a game instance, and run the game.
   ai = AlienInvasion()
   ai.run_game()
```