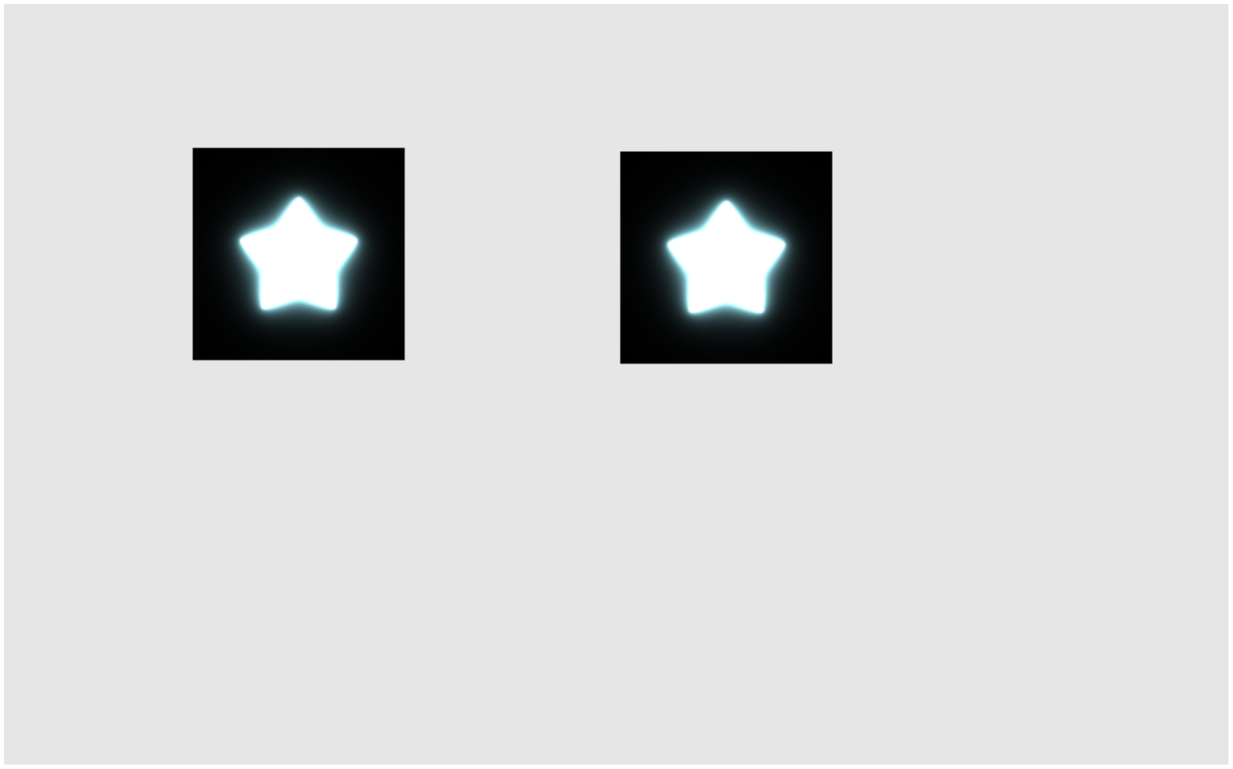
Alien Invasion (Chapter 13)

Create a seperate file in your alien invasion game for each of the following questions. Paste the content of the file into a Jupyter Notebook cell along with a screen shot of the game (2 cells per answer)

13-1. Stars: Find an image of a star. Make a grid of stars appear on the screen.

In [2]:
```python
# code
import pygame
from pygame.sprite import Sprite


class Star(Sprite):
    """Creating one single star."""

    def __init__(self, stars_game):
        """Initialize the star and set its starting position."""
        super().__init__()
        self.screen = stars_game.screen


        self.image = pygame.image.load('images/brightstar.png')
        self.rect = self.image.get_rect()

        # Start each new star near the top left of the screen.
        self.rect.x = self.rect.width
        self.rect.y = self.rect.height

        # Store the star's exact vertical position.
        self.y = float(self.rect.y)

import sys

import pygame

from settings import Settings
from star import Star


class StarsGame:
    """Managing star functions."""

    def __init__(self):
        """Initialize the game, and create game resources."""
        pygame.init()
        self.settings = Settings()

        self.screen = pygame.display.set_mode(
            (self.settings.screen_width, self.settings.screen_height))
        pygame.display.set_caption("Stars")

        self.stars = pygame.sprite.Group()
        self._create_stars()

    def run_game(self):
        """Start the main loop for the game."""
        while True:
            self._check_events()
            self._update_screen()

    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)

    def _check_keydown_events(self, event):
        """Respond to keypresses."""
        if event.key == pygame.K_q:
            sys.exit()

    def _create_stars(self):
        """Creating multiple stars."""
        star = Star(self)
        star_width, star_height = star.rect.size
        available_space_x = self.settings.screen_width - (star_width)
        number_stars_x = available_space_x // (2 * star_width)

        available_space_y = (self.settings.screen_height -
                             (2 * star_height))
        number_rows = available_space_y // (2 * star_height)

        # Fill the sky with stars.
        for row_number in range(number_rows):
            for star_number in range(number_stars_x):
                self._create_star(star_number, row_number)

    def _create_star(self, star_number, row_number):
        """Placement of star."""
        star = Star(self)
        star_width, star_height = star.rect.size
        star.x = star_width + 2 * star_width * star_number
        star.rect.x = star.x
        star.rect.y = star.rect.height + 2 * star.rect.height * row_number
        self.stars.add(star)

    def _update_screen(self):
        """Update images on the screen, and flip to the new screen."""
        self.screen.fill(self.settings.bg_color)
        self.stars.draw(self.screen)

        pygame.display.flip()


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


screen shot

13-2: Better Stars: You can make a more realistic star pattern by introducing randomness when you place each star. Recall that you can get a random number like this:

```
from random import randint
random_number = randint(-10,10)
```

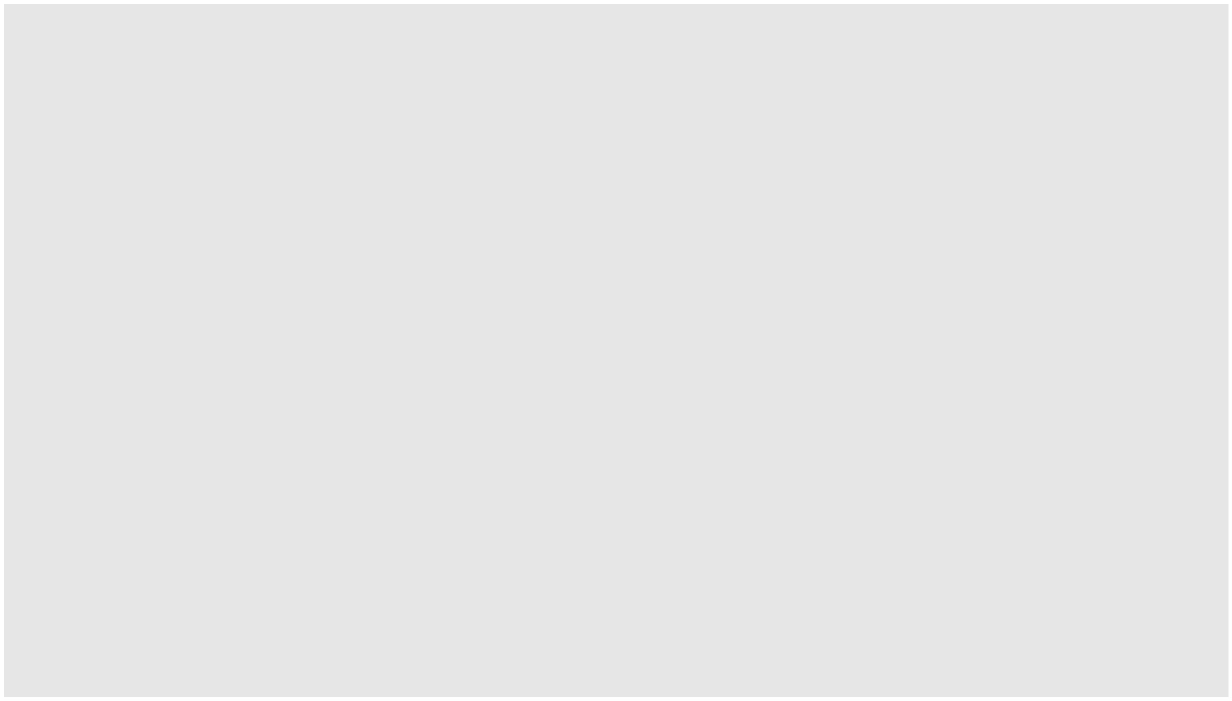
This code returns a random integer between -10 and 10. Using your code from 13-1, adjust each star's position by a random amount.

In [2]:
```
# code
import sys
import pygame

from settings import Settings
from star import Star

from random import randint


class StarsGame:
    """Managing star functions."""

    def __init__(self):
        """Initialize the game, and create game resources."""
        pygame.init()
        self.settings = Settings()

        self.screen = pygame.display.set_mode(
            (self.settings.screen_width, self.settings.screen_height))
        pygame.display.set_caption("Stars")

        self.stars = pygame.sprite.Group()
        self._create_stars()

    def run_game(self):
        """Start the main loop for the game."""
        while True:
            self._check_events()
            self._update_screen()

    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)

    def _check_keydown_events(self, event):
        """Respond to keypresses."""
        if event.key == pygame.K_q:
            sys.exit()

    def _create_stars(self):
        """Creating multiple stars."""
        star = Star(self)
        star_width, star_height = star.rect.size
        available_space_x = self.settings.screen_width - (star_width)
        number_stars_x = available_space_x // (2 * star_width)

        available_space_y = (self.settings.screen_height -
                             (2 * star_height))
        number_rows = available_space_y // (2 * star_height)

        # Fill the screen with stars.
        for row_number in range(number_rows):
            for star_number in range(number_stars_x):
                self._create_star(star_number, row_number)

    def _create_star(self, star_number, row_number):
        """Placement of star."""
        star = Star(self)
        star_width, star_height = star.rect.size
        star.x = star_width + 2 * star_width * star_number
        star.rect.x = star.x
        star.rect.y = star.rect.height + 2 * star.rect.height * row_number

        star.rect.x += randint(-7, 7)
        star.rect.y += randint(-7, 7)


        self.stars.add(star)

    def _update_screen(self):
        """Update images on the screen, and flip to the new screen."""
        self.screen.fill(self.settings.bg_color)
        self.stars.draw(self.screen)

        pygame.display.flip()


if __name__ == '__main__':
    sg = StarsGame()
    sg.run_game()
```

```
screen shot![randintstars.png](attachment:randintstars.png)
```

13-3. Raindrops: Find an image of a raindrop and create a grid of raindrops. Make the raindrops fall toward the bottom of the screen until they dissappear.

In [2]:
```
# code
import pygame
from pygame.sprite import Sprite


class Raindrop(Sprite):
    """A class to represent a single raindrop."""

    def __init__(self, rd_game):
        """Initialize the raindrop and set its starting position."""
        super().__init__()
        self.screen = rd_game.screen
        self.settings = rd_game.settings

        self.image = pygame.image.load('images/raindropgray.png')
        self.rect = self.image.get_rect()

        # horizontal
        self.rect.x = self.rect.width
        self.rect.y = self.rect.height

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

    def update(self):
        """Move the raindrop down the screen."""
        self.y += self.settings.raindrop_speed
        self.rect.y = self.y

import sys

import pygame

from settingsrain import Settings
from raindrop import Raindrop


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

    def __init__(self):
        """Initialize the game, and create game resources."""
        pygame.init()
        self.settings = Settings()

        self.screen = pygame.display.set_mode(
            (self.settings.screen_width, self.settings.screen_height))
        pygame.display.set_caption("Raindrops")

        self.raindrops = pygame.sprite.Group()
        self._create_drops()

    def run_game(self):
        """Start the main loop for the game."""
        while True:
            self._check_events()
            self.raindrops.update()
            self._update_screen()

    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)

    def _check_keydown_events(self, event):
        """Respond to keypresses."""
        if event.key == pygame.K_q:
            sys.exit()

    def _create_drops(self):
        """Fill the sky with raindrops."""

        drop = Raindrop(self)
        drop_width, drop_height = drop.rect.size
        available_space_x = self.settings.screen_width - drop_width
        number_drops_x = available_space_x // (2 * drop_width)

        available_space_y = self.settings.screen_height
        number_rows = available_space_y // (2 * drop_height)

        # Fill the screen with drops.
        for row_number in range(number_rows):
            for drop_number in range(number_drops_x):
                self._create_drop(drop_number, row_number)

    def _create_drop(self, drop_number, row_number):
        """Create an drop and place it in the row."""
        drop = Raindrop(self)
        drop_width, drop_height = drop.rect.size
        drop.rect.x = drop_width + 2 * drop_width * drop_number
        drop.y = 2 * drop.rect.height * row_number
        drop.rect.y = drop.y
        self.raindrops.add(drop)

    def _update_screen(self):
        """Update images on the screen, and flip to the new screen."""
        self.screen.fill(self.settings.bg_color)
        self.raindrops.draw(self.screen)

        pygame.display.flip()


if __name__ == '__main__':
    rd_game = RaindropsGame()
    rd_game.run_game()
```


screen shot

13-4. Steady Rain: Modify the code from 13-3 so when a row of rain drops disappears off the bottom of the screen, a new row appears at the top of the screen and begins to fall.

In [2]:
```
# code
import sys

import pygame

from settings import Settings
from raindrop import Raindrop

class RaindropsGame:
    """Manages functions of raindrops"""

    def __init__(self):
        """Initialize game"""
        pygame.init()
        self.settings = Settings()

        self.screen = pygame.display.set_mode(
                (self.settings.screen_width, self.settings.screen_height))
        pygame.display.set_caption("Raindrops")

        self.raindrops = pygame.sprite.Group()
        self._create_drops()

    def run_game(self):
        """Start the main loop for the game."""
        while True:
            self._check_events()
            self._update_raindrops()
            self._update_screen()

    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)

    def _check_keydown_events(self, event):
        """Respond to keypresses."""
        if event.key == pygame.K_q:
            sys.exit()

    def _create_drops(self):
        """Fill the screen with raindrops."""
        drop = Raindrop(self)
        drop_width, drop_height = drop.rect.size
        available_space_x = self.settings.screen_width - drop_width

        self.number_drops_x = available_space_x // (2 * drop_width)

        available_space_y = self.settings.screen_height
        number_rows = available_space_y // (2 * drop_height)

        # Fill the screen with drops.
        for row_number in range(number_rows):
                self._create_row(row_number)

    def _create_row(self, row_number):
        """single row"""
        for drop_number in range(self.number_drops_x):
            self._create_drop(drop_number, row_number)

    def _create_drop(self, drop_number, row_number):
        """placement of drop"""
        drop = Raindrop(self)
        drop_width, drop_height = drop.rect.size
        drop.rect.x = drop_width + 2 * drop_width * drop_number
        drop.y = 2 * drop.rect.height * row_number
        drop.rect.y = drop.y
        self.raindrops.add(drop)

    def _update_raindrops(self):
        """Showing drops that disappear"""
        self.raindrops.update()

        make_new_drops = False
        for drop in self.raindrops.copy():
            if drop.check_disappeared():
                # Remove this drop, and we'll need to make new drops.
                self.raindrops.remove(drop)
                make_new_drops = True

        # Make a new row of drops if needed.
        if make_new_drops:
            self._create_row(0)

    def _update_screen(self):
        """Update images on the screen, and flip to the new screen."""
        self.screen.fill(self.settings.bg_color)
        self.raindrops.draw(self.screen)

        pygame.display.flip()


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

screen shot

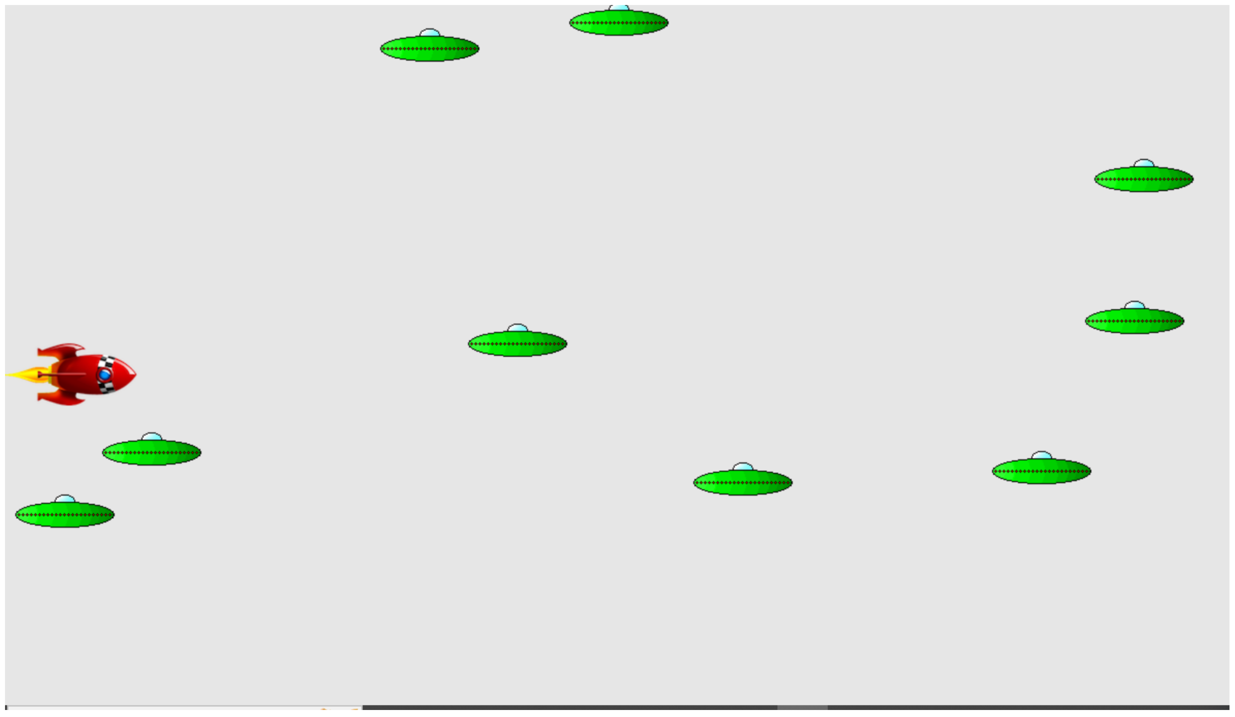
13-5 Sideways Shooter Part 2: We've come a long way since Exercise 12-6, Sideways Shooter. For this exercise, try to develop Sideways Shooter to the same point we've brought *Alien Invasion* to. Add a fleet of aliens, and make them move sideways toward the ship. Or, write code that places aliens at random positions along the right side of the screen and then sends them toward the ship. Also, write code that makes the aliens disappear when they're hit.

In [2]:
```
# code
import sys
from random import random

import pygame

from settingssideways2 import Settings
from shipalieninvasion2.py import Ship
from bulletalieninvasion2.py import Bullet
from alien import Alien

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

    def __init__(self):
        """Initialize the game, and create game resources."""
        pygame.init()
        self.settings = Settings()

        self.screen = pygame.display.set_mode(
                (self.settings.screen_width, self.settings.screen_height))
        pygame.display.set_caption("Sideways Shooter")

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

    def run_game(self):
        """Start the main loop for the game."""
        while True:
            self._check_events()

            # Consider creating a new alien.
            self._create_alien()


            self.ship.update()
            self._update_bullets()
            self.aliens.update()
            self._update_screen()

    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_UP:
            self.ship.moving_up = True
        elif event.key == pygame.K_DOWN:
```

```python
                self.ship.moving_down = True
            elif event.key == pygame.K_SPACE:
                self._fire_bullet()
            elif event.key == pygame.K_q:
                sys.exit()

        def _check_keyup_events(self, event):
            """Respond to key releases."""
            if event.key == pygame.K_UP:
                self.ship.moving_up = False
            elif event.key == pygame.K_DOWN:
                self.ship.moving_down = 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 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.left >= self.screen.get_rect().right:
                     self.bullets.remove(bullet)

            self._check_bullet_alien_collisions()

        def _check_bullet_alien_collisions(self):
            """Check whether any bullets have hit an alien."""
            collisions = pygame.sprite.groupcollide(
                    self.bullets, self.aliens, True, True)

        def _create_alien(self):
            """Create an alien, if conditions are right."""
            if random() < self.settings.alien_frequency:
                alien = Alien(self)
                self.aliens.add(alien)
                print(len(self.aliens))

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

            self.aliens.draw(self.screen)

            pygame.display.flip()


if __name__ == '__main__':
    # Make a game instance, and run the game.
        ss_game = SidewaysShooter()
        ss_game.run_game()

    # new file, settingssideways2.py
    class Settings:
        """A class to store all settings for Sideways Shooter."""

        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 = 3.0

            # Bullet settings
            self.bullet_speed = 6.0
            self.bullet_width = 15
            self.bullet_height = 3
            self.bullet_color = (60, 60, 60)
            self.bullets_allowed = 3

            self.alien_frequency = 0.008
            self.alien_speed = 1.5

    #new file, alien.py
    from random import randint

    import pygame
    from pygame.sprite import Sprite


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

        def __init__(self, ss_game):
            """Initialize the alien and set its starting position."""
            super().__init__()
            self.screen = ss_game.screen
            self.settings = ss_game.settings

            # Load the alien image and set its rect attribute.
            self.image = pygame.image.load('images/alienUFO.png')
            self.rect = self.image.get_rect()

            # Start each new alien at a random position on the right side
            #   of the screen.
            self.rect.left = self.screen.get_rect().right
            # The farthest down the screen we'll place the alien is the height
            #   of the screen, minus the height of the alien.
            alien_top_max = self.settings.screen_height - self.rect.height
            self.rect.top = randint(0, alien_top_max)

            # Store the alien's exact horizontal position.
            self.x = float(self.rect.x)
```

```python
    def update(self):
        """Move the alien steadily to the left."""
        self.x -= self.settings.alien_speed
        self.rect.x = self.x

#new file, bulletalieninvasion2.py
import pygame
from pygame.sprite import Sprite


class Bullet(Sprite):
    """A class to manage bullets fired from the ship."""

    def __init__(self, ss_game):
        """Create a bullet object at the ship's current position."""
        super().__init__()
        self.screen = ss_game.screen
        self.settings = ss_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.midright = ss_game.ship.rect.midright

        # Store the bullet's position as a decimal value.
        self.x = float(self.rect.x)

    def update(self):
        """Move the bullet across the screen."""
        # Update the decimal position of the bullet.
        self.x += self.settings.bullet_speed
        # Update the rect position.
        self.rect.x = self.x

    def draw_bullet(self):
        """Draw the bullet to the screen."""
        pygame.draw.rect(self.screen, self.color, self.rect)

# new file, shipalieninvasion2.py
import pygame

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

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

        # Load the ship image and get its rect.
        self.image = pygame.image.load('images/redrocket2.png')
        self.rect = self.image.get_rect()
        # Start each new ship at the center of the left side of the screen.
        self.rect.midleft = self.screen_rect.midleft

        # Store a decimal value for the ship's vertical position.
        self.y = float(self.rect.y)

        # Movement flags
        self.moving_up = False
        self.moving_down = False

    def update(self):
        """Update the ship's position based on movement flags."""
        # Update the ship's y value, not the rect.
        if self.moving_up and self.rect.top > 0:
            self.y -= self.settings.ship_speed
        if self.moving_down and self.rect.bottom < self.screen_rect.bottom:
            self.y += self.settings.ship_speed

        # Update rect object from self.y.
        self.rect.y = self.y

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

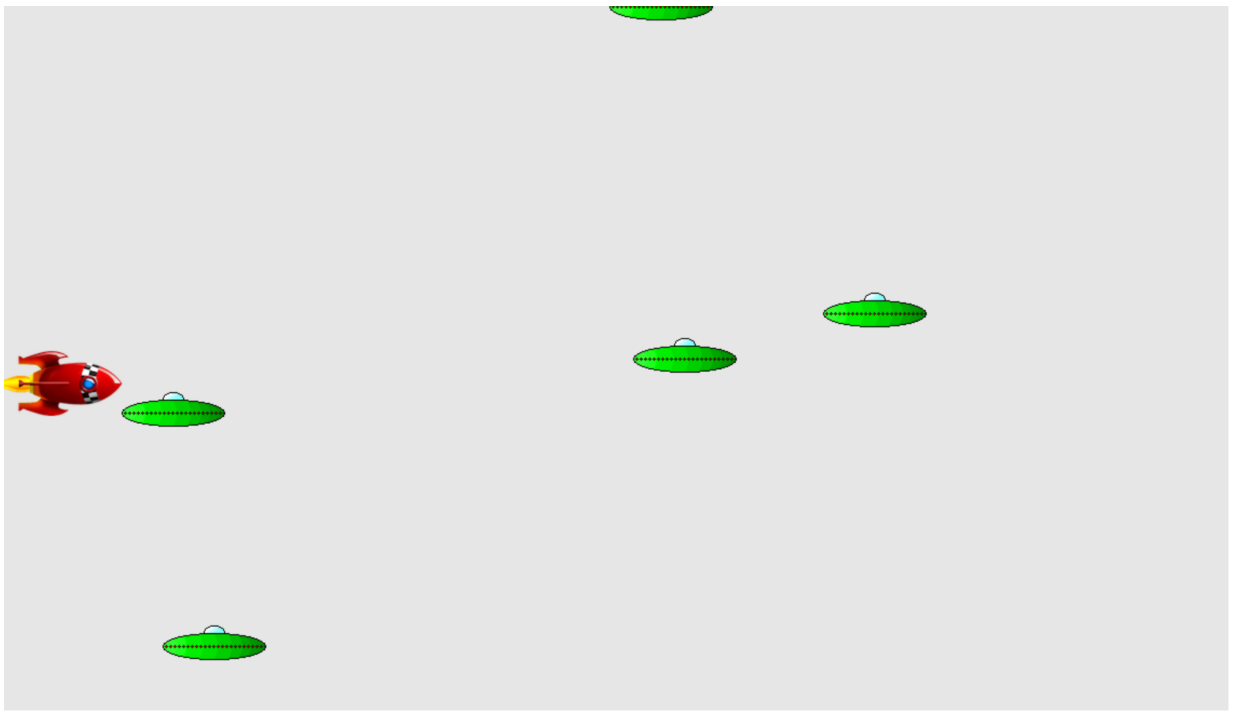
screen shot



13-6. Game Over: In Sideways Shooter, keep track of the number of times the ship is hit and the number of times an alien is hit by the ship. Decide on an appropriate condition for ending the game, and stop the game when this situation occurs.

In [2]:
```
# code
import sys
from random import random

import pygame

from settingssideways2 import Settings
from game_stats import GameStats
from shipalieninvasion3 import Ship
from bulletalieninvasion2 import Bullet
from alien import Alien


class SidewaysShooter:
    """Manages functions"""

    def __init__(self):
        """Initialize game."""
        pygame.init()
        self.settings = Settings()

        self.screen = pygame.display.set_mode(
            (self.settings.screen_width, self.settings.screen_height))
        pygame.display.set_caption("Sideways Shooter")

        # Create an instance to store game statistics.
        self.stats = GameStats(self)

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

    def run_game(self):
        """Start the main loop for the game."""
        while True:
            self._check_events()

            if self.stats.game_active:
                self._create_alien()

                self.ship.update()
                self._update_bullets()
                self._update_aliens()

            self._update_screen()

    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_UP:
            self.ship.moving_up = True
        elif event.key == pygame.K_DOWN:
            self.ship.moving_down = True
        elif event.key == pygame.K_SPACE:
            self._fire_bullet()
        elif event.key == pygame.K_q:
            sys.exit()

    def _check_keyup_events(self, event):
        """Respond to key releases."""
        if event.key == pygame.K_UP:
            self.ship.moving_up = False
        elif event.key == pygame.K_DOWN:
            self.ship.moving_down = False

    def _fire_bullet(self):
        """Creates new bullet."""
        if len(self.bullets) < self.settings.bullets_allowed:
            new_bullet = Bullet(self)
            self.bullets.add(new_bullet)

    def _update_bullets(self):
        """Update 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.left >= self.screen.get_rect().right:
                self.bullets.remove(bullet)

        self._check_bullet_alien_collisions()

    def _check_bullet_alien_collisions(self):
        """Check whether any bullets have hit an alien."""
        collisions = pygame.sprite.groupcollide(
            self.bullets, self.aliens, True, True)

    def _create_alien(self):
        """Create an alien, if conditions are right."""
        if random() < self.settings.alien_frequency:
            alien = Alien(self)
            self.aliens.add(alien)

    def _update_aliens(self):
        """Update alien positions, and look for collisions with ship."""
        self.aliens.update()

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

        # Look for aliens that have hit the left edge of the screen.
        self._check_aliens_left_edge()
    def _check_aliens_left_edge(self):
        """Same as ship getting hit."""

        for alien in self.aliens.sprites():
            if alien.rect.left < 0:
                self._ship_hit()
                break

    def _ship_hit(self):
        """Response"""
        if self.stats.ships_left > 0:
            # Decrement ships left.
            self.stats.ships_left -= 1

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

            # Center the ship.
            self.ship.center_ship()
        else:
            self.stats.game_active = False

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

        self.aliens.draw(self.screen)

        pygame.display.flip()


if __name__ == '__main__':
    # Make a game instance, and run the game.
    ss_game = SidewaysShooter()
    ss_game.run_game()


# new file, game stats
class GameStats:
    """Updates ."""

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

        # Start game in an active state.
        self.game_active = True

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

```
# shipalieninvasion3.py

import pygame


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

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

        # Load the ship image and get its rect.
        self.image = pygame.image.load('images/redrocket2.png')
        self.rect = self.image.get_rect()

        # Start each new ship at the center of the left side of the screen.
        self.center_ship()

        # Movement flags
        self.moving_up = False
        self.moving_down = False

    def update(self):
        """Update the ship's position based on movement flags."""
        # Update the ship's y value, not the rect.
        if self.moving_up and self.rect.top > 0:
            self.y -= self.settings.ship_speed
        if self.moving_down and self.rect.bottom < self.screen_rect.bottom:
            self.y += self.settings.ship_speed

        # Update rect object from self.y.
        self.rect.y = self.y
```



screen shot

```
In [ ]:
```