

# Laser Alien Collision Fix



I am making a space invaders game via pygame. I have an issue where I am trying to have the laser only die when hitting an active alien. Normally aliens have an explosion sequence before they die, so if I don't check for active aliens, the laser will get blocked by a dying alien. With this current check, it almost works. The laser will go through one alien, killing it, and hit another alien and killing it, then die. I just want it to only kill and die on one alien. Can you look at my code and help me?

Absolutely! I can help you sort this out. Please go ahead and paste the relevant parts of your code — especially the part where the laser and alien collision is handled. If you can include:

- The alien class (particularly how you track if an alien is "active")
- The laser class
- The part of the update loop or collision logic

That will help me get a full picture of what's going on and offer a fix.



Sure, I'll submit them 1 by 1

Perfect — send them over when you're ready, and I'll take a look at each part as it comes in.



```

Alien:
import pygame as pg
from vector import Vector
from point import Point
from laser import Laser
from pygame.sprite import Sprite
from timer import Timer
from random import randint

from colors import *

alien_image_scale = 5

class Alien(Sprite):
    alien_images0 = [pg.transform.scale_by(pg.image.load(f"images/space_invaders/alien0{n}.png"),
    alien_image_scale) for n in range(2)]
    alien_images1 = [pg.transform.scale_by(pg.image.load(f"images/space_invaders/alien1{n}.png"),
    alien_image_scale) for n in range(2)]
    alien_images2 = [pg.transform.scale_by(pg.image.load(f"images/space_invaders/alien2{n}.png"),

```

```

alien_image_scale) for n in range(2)]

    ufo_images0 = [[pg.transform.scale_by(pg.image.load(f"images/space_invaders/UFO.png"),
alien_image_scale)]]

    alien_images = [alien_images0, alien_images1, alien_images2]
    alien_boom = [pg.image.load(f"images/alien_boom{n}.png") for n in range(4)]

    alien_type = [{"color": "green", "points": 10, "alien_sprite": alien_images0, "alien_boom": :
alien_boom},
                  {"color": "cyan", "points": 20, "alien_sprite": alien_images1, "alien_boom": alien_boom},
                  {"color": "magenta", "points": 30, "alien_sprite": alien_images2, "alien_boom": :
alien_boom},
                  {"color": "red", "points": 100, "alien_sprite": alien_images0, "alien_boom": alien_boom} ]

def __init__(self, ai_game, v):
    super().__init__()
    self.ai_game = ai_game
    self.screen = ai_game.screen
    self.settings = ai_game.settings
    self.v = v
    self.is_active = True
    self.is_dying = False
    self.is_dead = False

    # Type to pick the alien images. Come back to later.
    type = randint(0, 2)
    self.timer = Timer(images=Alien.alien_images[type], delta=1000, start_index=type % 2)
    self.explosion_timer = Timer(images=Alien.alien_boom, loop_continuously=False,
running=False)
    self.image = self.timer.current_image()
    # print(self.image)
    self.rect = self.image.get_rect()

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

    self.x = float(self.rect.centerx)
    self.y = float(self.rect.centery)

    # Alien Guns
    self.guns = [self.rect.center]

    # Specify Laser Color
    self.laser_color = self.settings.alien_laser_color
    self.rng_laser_color = self.settings.rng_laser_color

    # Modified
    self.firedelay = 60
    self.ship_timer = 0

```

```

# Alien Laser -> Maybe we can append this to fleet lasers if necessary.
self.lasers = pg.sprite.Group()

# Firing?
self.firing = False

def hit(self):
    if not self.is_dying:
        print('ALIEN HIT! Alien is dying')
        self.is_dying = True
        self.is_active = False
        self.timer = self.explosion_timer
        self.timer.start()

def check_edges(self):
    sr = self.screen.get_rect()
    self.rect.centerx = self.x
    self.rect.centery = self.y
    r = self.rect
    # return self.x + self.rect.width >= sr.right or self.x <= 0
    return self.x + (self.rect.width / 2) >= sr.right or self.x <= (self.rect.width / 2)

def fire_laser(self, gun):
    laser = Laser(ai_game=self.ai_game, gun = gun, color = self.laser_color, v = Vector(0,-1),
                  width = self.settings.alien_laser_width, height = self.settings.alien_laser_height)
    self.lasers.add(laser)

def update(self):
    if self.is_dead: return
    if self.is_dying and self.explosion_timer.finished():
        self.is_dying = False
        self.is_dead = True
        print('Alien is dead')
        self.kill()
        return

    self.x += self.v.x
    self.y += self.v.y

    # Firing Test
    # Update gun location
    self.guns = [self.rect.center]

    # Modified Limits RoF
    if self.ship_timer % self.firedelay != 0:
        self.ship_timer += 1
    elif self.ship_timer % self.firedelay == 0:
        self.ship_timer == 0
        if self.firing:
            for gun in self.guns:
                if not self.is_dying: self.fire_laser(gun)

```

```

        self.ship_timer += 1

    # Update lasers
    self.lasers.update()
    for laser in self.lasers.copy():
        if laser.rect.bottom <= 0 or laser.rect.top >= self.settings.scr_height:
            self.lasers.remove(laser)
    for laser in self.lasers.sprites():
        laser.draw()

    self.image = self.timer.current_image()
    self.draw()

    def draw(self):
        self.rect.centerx = self.x
        self.rect.centery = self.y
        self.screen.blit(self.image, self.rect)

def main():
    print('\n run from alien_invasions.py\n')

if __name__ == "__main__":
    main()

```

Awesome, thanks for sharing the **Alien** class! It's super clear and modular — nicely done.

From what I can see, you're tracking alien states with:

- **is\_active** — for if it's alive and hittable
- **is\_dying** — explosion is happening
- **is\_dead** — already removed

The key part for your laser collision check will be to only allow collisions with aliens that are still **is\_active**. The problem you described sounds like your collision detection is looping over multiple active aliens and letting one laser hit more than one in a frame.

Now go ahead and send over the **Laser** class next, and then the **collision logic** where you check for collisions between lasers and aliens. That'll help me pinpoint exactly how to make the laser stop after one valid hit.



```

import pygame as pg
from pygame.sprite import Sprite
from vector import Vector

```

```

from random import randint

class Laser(Sprite):
    @staticmethod
    def random_color():
        return (randint(0, 255), randint(0, 255), randint(0, 255))
    def __init__(self, ai_game, color = (0,255,0), gun = (100, 100), v=Vector(), height = 5, width = 5):
        super().__init__()
        self.screen = ai_game.screen
        self.settings = ai_game.settings
        # self.color = self.settings.laser_color
        self.color = color
        self.height = height
        self.width = width
        self.gun = gun
        self.rect = pg.Rect(0, 0, width, height)
        self.rect.midtop = gun
        # self.rect.midtop = ai_game.ship.rect.midtop -> Replaced
        self.y = float(self.rect.centery)
        self.x = float(self.rect.centerx)
        # Initalizing Vector
        self.v = v

        # Continuous RNG Laser Toggle
        self.continuous_rng_laser = self.settings.continuous_rng_laser_color

        # # Laser Hit
        # self.laser_hit = False

        # # Additional Lasers -> Replaced
        # self.rect1 = pg.Rect(0, 0,
        #     self.settings.laser_width, self.settings.laser_height)
        # self.rect1.midtop = ai_game.ship.rect.midleft
        # self.y1 = float(self.rect1.y)

        # self.rect2 = pg.Rect(0, 0,
        #     self.settings.laser_width, self.settings.laser_height)
        # self.rect2.midtop = ai_game.ship.rect.midright
        # self.y2 = float(self.rect2.y)

        # def hit(self):
        #     laser_hit = True
        #     self.kill()

    def update(self):

        # if self.laser_hit: return

        self.y -= self.v.y * self.settings.laser_speed
        self.rect.centery = self.y

```

```

self.x -= self.v.x * self.settings.laser_speed
self.rect.centerx = self.x

if self.continuous_rng_laser: self.color = Laser.random_color()

# # Additional Lasers
# self.y1 -= self.settings.laser_speed
# self.rect1.y = self.y1

# self.y2 -= self.settings.laser_speed
# self.rect2.y = self.y2

def draw(self):
    pg.draw.rect(self.screen, self.color, self.rect)
    # pg.draw.rect(self.screen, self.color, self.rect1)
    # pg.draw.rect(self.screen, self.color, self.rect2)

def main():
    print("\nYou have to run from alien_invasion.py\n")

if __name__ == "__main__":
    main()

```

Thanks! The **Laser** class looks solid. I see you were already thinking about handling hits with that commented-out **laser\_hit** flag, which is smart.

Now, the final piece I need to give you the fix is the **collision logic** between the laser and alien — probably something like:

```

python

collisions = pg.sprite.groupcollide(lasers, aliens, True, False)

```

Or however you're currently handling that in your game loop. Please paste that part next — especially the logic where the laser interacts with aliens, so I can modify it to ensure **one laser only hits one active alien and stops**, even if multiple are close together.

You're super close to the fix already!



Before we get to this, one thing you should know is that the original code had ship updating some of the lasers. I might change this, but for now just here it is:

```

import pygame as pg
from vector import Vector
from point import Point
from laser import Laser
from timer import Timer
from time import sleep
from pygame.sprite import Sprite

```

```

from random import randint

class Ship(Sprite):
    def __init__(self, ai_game, v=Vector()):
        super().__init__()
        self.ai_game = ai_game
        self.screen = ai_game.screen
        self.settings = ai_game.settings
        self.screen_rect = ai_game.screen.get_rect()
        self.stats = ai_game.stats
        self.sb = None

        self.image = pg.image.load('images/ship.bmp')
        self.rect = self.image.get_rect()
        self.rect.midbottom = self.screen_rect.midbottom
        scr_r = self.screen_rect
        self.x = float(scr_r.midbottom[0])
        self.y = float(scr_r.height)
        self.v = v
        self.lasers = pg.sprite.Group()
        self.firing = False
        self.fleet = None

        # Separate guns
        self.guns = [self.rect.midtop, self.rect.midleft, self.rect.midright]

        # Specify Laser Color
        self.laser_color = self.settings.laser_color
        self.rng_laser_color = self.settings.rng_laser_color

        # Modified
        self.firedelay = 1
        self.ship_timer = 0

        # Death Flags
        self.is_dying = False

        self.ship_boom = [pg.image.load(f"images/alien_boom{n}.png") for n in range(4)]
        self.explosion_timer = Timer(images=self.ship_boom, loop_continuously=False, running=False)

    def set_fleet(self, fleet): self.fleet = fleet

    def set_sb(self, sb): self.sb = sb

    def reset_ship(self):
        self.lasers.empty()
        self.center_ship()

    def center_ship(self):
        self.rect.midbottom = self.screen_rect.midbottom
        self.x = float(self.rect.centerx)

```

```

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

def bound(self):
    x, y, scr_r = self.x, self.y, self.screen_rect
    # self.x = max(0, min(x, scr_r.width - self.rect.width))
    # self.y = max(0, min(y, scr_r.height - self.rect.height))
    self.x = max(self.rect.width / 2, min(x, scr_r.width - (self.rect.width / 2)))
    self.y = max(self.rect.height / 2, min(y, scr_r.height - (self.rect.height / 2)))

def ship_hit(self):

    if not self.is_dying:
        print('SHIP HIT! SHIP is dying')
        self.is_dying = True
        self.timer = self.explosion_timer
        self.timer.start()

    self.stats.ships_left -= 1
    print(f"Only {self.stats.ships_left} ships left now")
    self.sb.prep_ships()
    if self.stats.ships_left <= 0:
        self.ai_game.game_over()

    self.lasers.empty()
    self.fleet.aliens.empty()

    self.center_ship()
    self.fleet.create_fleet()

    sleep(0.5)

def fire_laser(self, gun):
    laser = Laser(ai_game=self.ai_game, gun = gun, color = self.laser_color, v = Vector(0,1),
                  width = self.settings.laser_width, height = self.settings.laser_height)
    self.lasers.add(laser)

def open_fire(self): self.firing = True

def cease_fire(self): self.firing = False

def single_fire(self):
    self.fire_laser(self.rect.midtop)

def update(self):
    self.x += self.v.x
    self.y += self.v.y
    self.bound()

    # Modified - Separate Guns
    self.guns = [self.rect.midtop, self.rect.midleft, self.rect.midright]

```



```
# Random Laser Color
if self.rng_laser_color: self.laser_color = (randint(0, 255), randint(0, 255), randint(0, 255))

# Modified Limits RoF
if self.ship_timer % self.firedelay != 0:
    self.ship_timer += 1
elif self.ship_timer % self.firedelay == 0:
    self.ship_timer == 0
    if self.firing:
        for gun in self.guns:
            self.fire_laser(gun)
        self.ship_timer += 1

# Update Lasers
self.lasers.update()
for laser in self.lasers.copy():
    if laser.rect.bottom <= 0 or laser.rect.top >= self.settings.scr_height:
        self.lasers.remove(laser)
for laser in self.lasers.sprites():
    laser.draw()

self.draw()

def draw(self):
    self.rect.centerx, self.rect.centery = self.x, self.y
    self.screen.blit(self.image, self.rect)

def main():
    print('\n*** message from ship.py --- run from alien_invasions.py\n')

if __name__ == "__main__":
    main()
```

Thanks for sharing the **Ship** class — everything is fitting together really nicely. You've got well-organized logic and clean laser control. Now that I've seen:

- **Alien** class with the **is\_active**, **is\_dying**, **is\_dead** flags,
- **Laser** class which doesn't currently self-kill on hit (but could),
- and the **Ship** firing logic,

The **final piece** is how and **where you check for laser-alien collisions**. That's where we'll need to:

1. Ensure a laser only affects **one** active alien.
2. Stop checking collisions for a laser after it hits an alien.
3. Prevent lasers from interacting with **is\_dying** aliens.

If you can paste **the part of your game loop or update logic where this collision check happens**, I can help you rewrite that section to behave exactly how you want — i.e., the laser should:

- Only hit aliens that are `is_active`.
- Only hit **one** alien.
- Kill itself after hitting the first valid alien.

It's likely happening in `alien_invasion.py`, in the `update()` or `check_collisions()` method. Could you share that next?



```
import pygame as pg
from vector import Vector
from point import Point
from laser import Laser

from alien import Alien
from pygame.sprite import Sprite

class Fleet(Sprite):
    def __init__(self, ai_game):
        self.ai_game = ai_game
        self.screen = ai_game.screen
        self.ship = ai_game.ship
        self.aliens = pg.sprite.Group()
        self.settings = ai_game.settings
        self.stats = ai_game.stats
        self.sb = ai_game.sb
        self.v = Vector(self.settings.alien_speed, 0)
        # alien = Alien(ai_game=ai_game)
        # self.aliens.add(alien)
        self.spacing = 1.4
        self.create_fleet()
        # self.create_row()

        self.lasers = pg.sprite.Group()

    def reset_fleet(self):
        self.aliens.empty()
        self.create_fleet()

    def create_fleet(self):
        """Create a fleet of aliens with a fixed number of columns."""
        self.v = Vector(self.settings.alien_speed, 0)

        # Set the desired number of columns and rows
        num_columns = self.settings.alien_columns
        num_rows = self.settings.alien_rows

        # Get the size of the largest alien sprite for spacing
        alien_tile = Alien.alien_images2[0].get_rect()
        tile_width = alien_tile.width
```

```

tile_height = alien_tile.height

# Calculate horizontal spacing
column_spacing = tile_width * self.spacing

# Create the fleet using specified columns and calculated rows
for row in range(int(num_rows)):
    y = (row + 1) * tile_height * self.spacing
    self.create_row(y, num_columns, column_spacing)

def create_row(self, y, num_columns, column_spacing):
    """Create a row of aliens with a fixed number of columns."""
    alien_tile = Alien.alien_images2[0].get_rect()
    tile_width = alien_tile.width

    for col in range(num_columns):
        x = (col + 1) * tile_width * self.spacing
        new_alien = Alien(self, v=self.v)
        new_alien.rect.centery = y
        new_alien.y = y
        new_alien.x = x
        new_alien.rect.centerx = x
        self.aliens.add(new_alien)

# def create_fleet(self):
#     # Updating velocity
#     self.v = Vector(self.settings.alien_speed, 0)

#     # Creating fleet
#     alien = Alien(ai_game=self.ai_game, v=self.v)

#     # Grabbing largest alien for spacing
#     alien_tile = Alien.alien_images2[0].get_rect()

#     tile_height = alien_tile.height
#     current_y = tile_height
#     while current_y < (self.settings.scr_height - self.spacing * 6 * tile_height):
#         self.create_row(current_y)
#         current_y += self.spacing * tile_height

# def create_row(self, y):
#     alien = Alien(ai_game=self.ai_game, v=self.v)
#     # tile_width = alien.rect.width

#     # Grabbing largest alien for spacing
#     alien_tile = Alien.alien_images2[0].get_rect()

#     tile_width = alien_tile.width
#     current_x = tile_width
#     while current_x < (self.settings.scr_width - self.spacing * tile_width):
#         new_alien = Alien(self, v=self.v)

```

```
#     new_alien.rect.centery = y
#     new_alien.y = y
#     new_alien.x = current_x
#     new_alien.rect.centerx = current_x
#     self.aliens.add(new_alien)
#     current_x += self.spacing * tile_width

def check_edges(self):
    for alien in self.aliens:
        if alien.check_edges():
            return True
    return False

def check_bottom(self):
    for alien in self.aliens:
        if alien.rect.bottom >= self.settings.scr_height:
            self.ship.ship_hit()
            return True
    return False

def update(self):
    collisions = pg.sprite.groupcollide(self.ship.lasers, self.aliens, False, False)

    print(collisions)

    if collisions:

        for lasers, aliens in collisions.items():
            self.stats.score += self.settings.alien_points * len(aliens)
            for alien in aliens:
                if alien.is_active:
                    lasers.kill()
                    alien.hit()

        self.sb.prep_score()
        self.sb.check_high_score()

    if not self.aliens:
        self.ship.lasers.empty()

        self.settings.increase_speed()
        self.create_fleet()
        # Increase level.
        self.stats.level += 1
        self.sb.prep_level()
        return

    # Check if aliens hit the ship.
    if pg.sprite.spritecollideany(self.ship, self.aliens):
        print("Ship hit!")
        self.ship.ship_hit()
```

```

return

# Check if alien lasers hit the ship
for self.alien in self.aliens:
    if pg.sprite.spritecollideany(self.ship, self.alien.lasers):
        print("Ship hit!")
        self.ship.ship_hit()
        return

# Check if aliens reached the bottom
if self.check_bottom():
    return

# Check if an alien has hit the edge
if self.check_edges():
    self.v.x *= -1
    for alien in self.aliens:
        alien.v.x = self.v.x
        alien.y += self.settings.fleet_drop_speed

for alien in self.aliens:
    alien.update()

def draw(self): pass
# for alien in self.aliens:
#     alien.draw()

def main():
    print('\n run from alien_invasions.py\n')

if __name__ == "__main__":
    main()

```

Awesome — thanks for sharing the `Fleet` class. The collision check is right here:

```

python

collisions = pg.sprite.groupcollide(self.ship.lasers, self.aliens, False, False)

```

The current issue is that **groupcollide returns all overlapping laser-alien pairs**, even if the alien is not active. Plus, if a laser overlaps multiple aliens in the same frame (could happen with close spacing), it'll affect all of them — not just the first one. So, let's fix that.

### ✓ Here's what you want:

- Only hit active (`is_active`) aliens
- Each laser should only hit *one* alien
- Only then should the alien `.hit()` and the laser `.kill()`

## ✓ Modified Collision Code:

Replace this section in your `update()` method:

```
python

collisions = pg.sprite.groupcollide(self.ship.lasers, self.aliens, False, False)

print(collisions)

if collisions:
    for lasers, aliens in collisions.items():
        self.stats.score += self.settings.alien_points * len(aliens)
        for alien in aliens:
            if alien.is_active:
                lasers.kill()
                alien.hit()

self.sb.prep_score()
self.sb.check_high_score()
```

With this improved version:

```
python

for laser in self.ship.lasers.copy():
    for alien in self.aliens:
        if alien.is_active and laser.rect.colliderect(alien.rect):
            alien.hit()
            laser.kill()
            self.stats.score += self.settings.alien_points
            self.sb.prep_score()
            self.sb.check_high_score()
            break # Laser hit one alien, don't check others
```

## 🔍 Why This Works Better:

- It manually checks each laser against aliens using `colliderect`, so you have more control.
- It filters **only** `is_active` aliens.
- Once a laser hits **one alien**, it breaks out of the loop — no multi-hits.
- Keeps the scoring and scoreboard update working exactly as before.

Let me know if you want to animate the laser disappearing or hook this up with sound effects later. You're super close to Space Invaders perfection now. 🚀



It was a good effort, but for some reason, it will still hit one more alien before dying!

Ah — I see what's happening now! That's a subtle but *very* important detail. Here's what's likely going wrong:

Even though we call `laser.kill()`, the laser is still being processed in the current frame — so it **can collide with a second alien before it's actually removed from the group**.

This is a known quirk with `sprite.kill()` inside loops — it marks the sprite for removal but doesn't remove it from memory *immediately*. So if the loop continues, that "dead" laser is still colliding.

## Fix Options

Here are two rock-solid solutions — pick your favorite:

### ✓ Option 1: Use a flag to stop further collisions

Instead of relying on `.kill()` to stop things, track it manually inside the loop.

```
python

for laser in self.ship.lasers.copy():
    laser_hit = False
    for alien in self.aliens:
        if laser_hit:
            break # Laser already hit something

        if alien.is_active and laser.rect.colliderect(alien.rect):
            alien.hit()
            laser.kill()
            laser_hit = True # Set flag to break outer loop logic
            self.stats.score += self.settings.alien_points
            self.sb.prep_score()
            self.sb.check_high_score()
```

This guarantees one laser hits one alien, **and then stops**.

### ✓ Option 2: Filter aliens using `sprite.spritecollide()` with `dokill=True`

This uses Pygame's built-in `spritecollide()` with a custom collision filter:

```
python

for laser in self.ship.lasers.copy():
    collided_aliens = [alien for alien in self.aliens if alien.is_active and
laser.rect.colliderect(alien.rect)]
    if collided_aliens:
        alien = collided_aliens[0] # Only hit the first one
        alien.hit()
        laser.kill()
        self.stats.score += self.settings.alien_points
        self.sb.prep_score()
        self.sb.check_high_score()
```

Same result — just a little more Pythonic with filtering.

## Why This Happens

`.kill()` removes the sprite from all groups **on the next group update cycle**, but inside the current loop, it still exists in memory. So without a flag or loop break, your laser gets a chance to collide again.

Let me know which option you like best — or if you want to log laser IDs for debugging. You're right on the edge of pixel-perfect behavior 🚀💥



Two still die in both. Regardless of the size of the laser or the space between aliens. I am starting to think the cause could lie elsewhere, but i am not sure where to look.

Got it — and if it's *always* hitting **two aliens**, even when spaced far apart and regardless of laser size, that strongly suggests the problem lies **outside the collision loop logic itself**.

Let's dig deep. 🔍

## 🧠 Let's Understand the Clues

### What we know:

- One laser causes **two aliens** to die on impact.
- `laser.kill()` is called.
- Laser is added to `self.ship.lasers`, and `Fleet.update()` handles the collisions.
- `alien.hit()` is called and sets `alien.is_active = False`.
- Each alien is being hit *only once* (presumably).
- You've tried spacing out aliens — same result.

## ✅ Let's Trace Possible Root Causes

### 1. Multiple Aliens Sharing the Same Rect

If you're reusing the same alien object (or rect) more than once, one laser could appear to hit multiple clones. Check this in your `create_row` or `create_fleet` logic:

```
python
new_alien = Alien(self, v=self.v)
```

Make sure **each alien instance** is unique and independent — which it looks like it is in your code.

Still, add this debug print inside `create_row()`:

```
python
print(f"Creating alien at x={x}, y={y}, id={id(new_alien)}")
```

Then compare the `ids`. If two aliens have the same `id`, they're the same object, reused by mistake.

### 2. `alien.hit()` Not Disabling Alien Immediately



If `alien.hit()` doesn't **actually disable the alien right away**, it could still collide again in the same frame. Check your `hit()` method in `alien.py`. You might have:

```
python

def hit(self):
    self.is_active = False
    self.timer.start()
```

But is that alien still being drawn or counted in collisions **before** `is_active` is checked again?

Add debug output inside the `Alien.hit()` method:

```
python

def hit(self):
    print(f"Alien hit: id={id(self)} is_active={self.is_active}")
    if not self.is_active:
        return
    self.is_active = False
    self.timer.start()
```

Make sure that once hit, it **really stops** interacting.

### 3. Laser Group Not Updating Properly After `kill()`

Let's try this in `Fleet.update()`:

```
python

collisions = pg.sprite.groupcollide(self.ship.lasers, self.aliens, False, False)

if collisions:
    for laser, hit.aliens in collisions.items():
        hit_count = 0
        for alien in hit.aliens:
            if alien.is_active and hit_count == 0:
                alien.hit()
                laser.kill()
                hit_count += 1
                self.stats.score += self.settings.alien_points
                self.sb.prep_score()
                self.sb.check_high_score()
```

The `hit_count == 0` ensures **only one alien is hit per laser**, even if Pygame for some reason says it collided with multiple.

If that **still hits two**, something's deeply wrong, and I'd ask:

### 4. Is the Laser Somehow Being Spawned Twice?

This is a **real possibility**, especially since you're doing a `for gun in self.guns` loop inside `Ship.update()`:

```
python

for gun in self.guns:
    self.fire_laser(gun)
```

👁 You might be firing **multiple lasers per frame**, and **two overlap perfectly** — giving the illusion of *one laser*, but really two are flying together.

Let's test this. Inside `Laser.__init__`, add:

```
python

print(f"Laser created at {gun}, id={id(self)}")
```

Now fire once and see how many lasers get created — especially when `firing = True`.

If you see **multiple lasers created per fire** (even from the same spot), **that's your problem**.

## ✅ Final Step — Try This Sanity Check

Temporarily change your `Ship.single_fire()` to fire **only once**, and comment out all auto-fire:

```
python

def update(self):
    self.x += self.v.x
    self.y += self.v.y
    self.bound()

    self.guns = [self.rect.midtop] # Only one gun for test
    if self.firing:
        self.fire_laser(self.guns[0])
        self.firing = False # Disable continuous fire for test

    self.lasers.update()
    for laser in self.lasers.copy():
        if laser.rect.bottom <= 0 or laser.rect.top >= self.settings.scr_height:
            self.lasers.remove(laser)
    for laser in self.lasers.sprites():
        laser.draw()

    self.draw()
```

Now hit space, fire one laser, and see if **only one alien dies**.

## TL;DR

- Suspect: multiple lasers stacking, not one laser hitting multiple aliens.
- Print laser creation info (`id(self)`), and see if you're making more than one.
- Use a one-laser test case to verify.

Want to paste your `fire_laser()` or `open_fire()` logic? I can help pinpoint it exactly.



For # 1

Creating alien at x=84.0, y=56.0, id=1710468005040

Creating alien at x=168.0, y=56.0, id=1710469333072

Creating alien at x=251.99999999999997, y=56.0, id=1710469333392  
 Creating alien at x=336.0, y=56.0, id=1710468151936  
 Creating alien at x=420.0, y=56.0, id=1710468152240  
 Creating alien at x=503.99999999999994, y=56.0, id=1710469316976  
 Creating alien at x=588.0, y=56.0, id=1710468674384  
 Creating alien at x=672.0, y=56.0, id=1710468674656  
 Creating alien at x=756.0, y=56.0, id=1710469091920  
 Creating alien at x=84.0, y=112.0, id=1710469092176  
 Creating alien at x=168.0, y=112.0, id=1710469092432  
 Creating alien at x=251.99999999999997, y=112.0, id=1710469092688  
 Creating alien at x=336.0, y=112.0, id=1710469092944  
 Creating alien at x=420.0, y=112.0, id=1710469093200  
 Creating alien at x=503.99999999999994, y=112.0, id=1710469093456  
 Creating alien at x=588.0, y=112.0, id=1710469093712  
 Creating alien at x=672.0, y=112.0, id=1710469093968  
 Creating alien at x=756.0, y=112.0, id=1710469094224  
 Creating alien at x=84.0, y=168.0, id=1710469094480  
 Creating alien at x=168.0, y=168.0, id=1710469094736  
 Creating alien at x=251.99999999999997, y=168.0, id=1710469094992  
 Creating alien at x=336.0, y=168.0, id=1710469095248  
 Creating alien at x=420.0, y=168.0, id=1710469095504  
 Creating alien at x=503.99999999999994, y=168.0, id=1710469095760  
 Creating alien at x=588.0, y=168.0, id=1710469096016  
 Creating alien at x=672.0, y=168.0, id=1710469096272  
 Creating alien at x=756.0, y=168.0, id=1710469096528  
 Creating alien at x=84.0, y=224.0, id=1710469096784  
 Creating alien at x=168.0, y=224.0, id=1710469097040  
 Creating alien at x=251.99999999999997, y=224.0, id=1710469097296  
 Creating alien at x=336.0, y=224.0, id=1710469097552  
 Creating alien at x=420.0, y=224.0, id=1710469097808  
 Creating alien at x=503.99999999999994, y=224.0, id=1710469098064  
 Creating alien at x=588.0, y=224.0, id=1710469098320  
 Creating alien at x=672.0, y=224.0, id=1710469098576  
 Creating alien at x=756.0, y=224.0, id=1710469098832  
 Creating alien at x=84.0, y=280.0, id=1710469099088  
 Creating alien at x=168.0, y=280.0, id=1710469099344  
 Creating alien at x=251.99999999999997, y=280.0, id=1710469099600  
 Creating alien at x=336.0, y=280.0, id=1710469099856  
 Creating alien at x=420.0, y=280.0, id=1710469100112  
 Creating alien at x=503.99999999999994, y=280.0, id=1710469100368  
 Creating alien at x=588.0, y=280.0, id=1710469100624  
 Creating alien at x=672.0, y=280.0, id=1710469100880  
 Creating alien at x=756.0, y=280.0, id=1710469101136  
 Creating alien at x=84.0, y=336.0, id=1710469101392  
 Creating alien at x=168.0, y=336.0, id=1710469101648  
 Creating alien at x=251.99999999999997, y=336.0, id=1710469101904  
 Creating alien at x=336.0, y=336.0, id=1710469102160  
 Creating alien at x=420.0, y=336.0, id=1710469102416  
 Creating alien at x=503.99999999999994, y=336.0, id=1710469102672  
 Creating alien at x=588.0, y=336.0, id=1710469102928  
 Creating alien at x=672.0, y=336.0, id=1710469103184


Creating alien at x=756.0, y=336.0, id=1710469349456  
 Creating alien at x=84.0, y=56.0, id=1710469091408  
 Creating alien at x=168.0, y=56.0, id=1710469091152  
 Creating alien at x=251.99999999999997, y=56.0, id=1710469090640  
 Creating alien at x=336.0, y=56.0, id=1710469091664  
 Creating alien at x=420.0, y=56.0, id=1710469091920  
 Creating alien at x=503.99999999999994, y=56.0, id=1710469092176  
 Creating alien at x=588.0, y=56.0, id=1710469092432  
 Creating alien at x=672.0, y=56.0, id=1710469092688  
 Creating alien at x=756.0, y=56.0, id=1710469092944  
 Creating alien at x=84.0, y=112.0, id=1710469093200  
 Creating alien at x=168.0, y=112.0, id=1710469093456  
 Creating alien at x=251.99999999999997, y=112.0, id=1710469093712  
 Creating alien at x=336.0, y=112.0, id=1710469093968  
 Creating alien at x=420.0, y=112.0, id=1710469094224  
 Creating alien at x=503.99999999999994, y=112.0, id=1710469094480  
 Creating alien at x=588.0, y=112.0, id=1710469094736  
 Creating alien at x=672.0, y=112.0, id=1710469094992  
 Creating alien at x=756.0, y=112.0, id=1710469095248  
 Creating alien at x=84.0, y=168.0, id=1710469095504  
 Creating alien at x=168.0, y=168.0, id=1710469095760  
 Creating alien at x=251.99999999999997, y=168.0, id=1710469096016  
 Creating alien at x=336.0, y=168.0, id=1710469096272  
 Creating alien at x=420.0, y=168.0, id=1710469096528  
 Creating alien at x=503.99999999999994, y=168.0, id=1710469096784  
 Creating alien at x=588.0, y=168.0, id=1710469097040  
 Creating alien at x=672.0, y=168.0, id=1710469097296  
 Creating alien at x=756.0, y=168.0, id=1710469097552  
 Creating alien at x=84.0, y=224.0, id=1710469097808  
 Creating alien at x=168.0, y=224.0, id=1710469098064  
 Creating alien at x=251.99999999999997, y=224.0, id=1710469098320  
 Creating alien at x=336.0, y=224.0, id=1710469098576  
 Creating alien at x=420.0, y=224.0, id=1710469098832  
 Creating alien at x=503.99999999999994, y=224.0, id=1710469099088  
 Creating alien at x=588.0, y=224.0, id=1710469099344  
 Creating alien at x=672.0, y=224.0, id=1710469099600  
 Creating alien at x=756.0, y=224.0, id=1710469099856  
 Creating alien at x=84.0, y=280.0, id=1710469100112  
 Creating alien at x=168.0, y=280.0, id=1710469100368  
 Creating alien at x=251.99999999999997, y=280.0, id=1710469100624  
 Creating alien at x=336.0, y=280.0, id=1710469100880  
 Creating alien at x=420.0, y=280.0, id=1710469101136  
 Creating alien at x=503.99999999999994, y=280.0, id=1710469101392  
 Creating alien at x=588.0, y=280.0, id=1710469101648  
 Creating alien at x=672.0, y=280.0, id=1710469101904  
 Creating alien at x=756.0, y=280.0, id=1710469102160  
 Creating alien at x=84.0, y=336.0, id=1710469102416  
 Creating alien at x=168.0, y=336.0, id=1710469102672  
 Creating alien at x=251.99999999999997, y=336.0, id=1710469102928  
 Creating alien at x=336.0, y=336.0, id=1710469103184  
 Creating alien at x=420.0, y=336.0, id=1710469349712

```

Creating alien at x=503.99999999999994, y=336.0, id=1710469349968
Creating alien at x=588.0, y=336.0, id=1710469350224
Creating alien at x=672.0, y=336.0, id=1710469350480
Creating alien at x=756.0, y=336.0, id=1710469350736

```

I'll check #2

Great! And based on that log from #1 — you're **not** reusing any alien objects.  Every alien has a unique ID, which rules out shared instances or reused rects. That's one mystery crossed off the list.

Now onto #2... and possibly the biggest clue of all. While you're checking `alien.hit()` behavior, here's one more **super likely cause** we should test alongside that:

## THEORY: Laser Is Still Active During Same Frame

Let's suppose this is your collision logic:

```

python

collisions = pg.sprite.groupcollide(self.ship.lasers, self.aliens, False, False)

if collisions:
    for laser, hit.aliens in collisions.items():
        for alien in hit.aliens:
            if alien.is_active:
                alien.hit()
                laser.kill()

```

Even though you're calling `laser.kill()` inside the loop, Pygame **may still detect that laser as colliding with multiple aliens in the same `groupcollide` call** — *before* it's removed from the group.

This means:

- Collision is calculated *before* `laser.kill()`
- All detected collisions are returned in `collisions`
- So even though you kill the laser on the first alien, you're *still looping through more aliens it's already marked as hit*

This matches exactly what you're seeing: **one laser always hits two aliens.**

## FIX: Limit Laser to One Hit

Here's how you can enforce one-hit-per-laser protection *even if `groupcollide` returns multiple hits*:

 Replace your laser collision loop with this:

```

python

collisions = pg.sprite.groupcollide(self.ship.lasers, self.aliens, False, False)

if collisions:
    for laser, hit.aliens in collisions.items():

```

```

for alien in hit.aliens:
    if alien.is_active:
        alien.hit()
        laser.kill() # Prevent future hits
        self.stats.score += self.settings.alien_points
        self.sb.prep_score()
        self.sb.check_high_score()
        break # 🌟 Stop after hitting ONE alien

```

The key is that **break** — it exits the `for alien in hit.aliens` loop after the first valid hit, **before any second alien can be affected**.

---

## 🔍 Also: Double-check `alien.hit()` logic

Here's a minimal version of what that should look like:

```

python

def hit(self):
    if not self.is_active:
        return
    self.is_active = False
    self.timer.start()
    print(f"Alien {id(self)} hit.")

```

Make sure the alien really becomes inactive immediately. If you're calling `hit()` multiple times, the log will show it.

---

## Want Extra Clarity?

You can add this just before `laser.kill()`:

```

python

print(f"Laser {id(laser)} hit alien {id(alien)} at ({alien.rect.x}, {alien.rect.y})")

```

See if **that same laser ID** is hitting multiple aliens in the same frame.

---

Let me know what you find with #2, and try that **break** in the loop — I'm betting it fixes the two-death mystery!



Alright, so with your `def hit`, the aliens wouldn't die cause we removed the normal sequence, so I'll revert. BUUUUUUUUUUUUUUT I found something interesting.

```

Alien 2331835417168 hit.
Laser 2331834088624 hit alien 2331835417168 at (577, 316)
Alien 2331835184976 hit.
Laser 2331835367504 hit alien 2331835184976 at (578, 260)

```

OHHHHH snap — that is *super* interesting! 🧐

Two different **lasers**, hitting two different **aliens**... and those aliens are *stacked vertically* —  $y=316$  and  $y=260$ . So what we're seeing is:

- **Laser A** hits Alien A ✅
- **Laser B** hits Alien B ✅
- But they're *so close together*, it looks like **one** laser is causing **two** hits visually, when in fact...
- There are actually **two lasers** doing the damage in the same frame!

## 🧠 Here's What Might Be Happening:

When you press fire rapidly or have multiple lasers on screen:

- They're traveling vertically,
- And they're *overlapping*, or at least super close in x/y space,
- So both hit in the same frame (especially if aliens are tightly packed),
- Creating the **illusion** that one laser hit two aliens.

This is totally possible if your firing delay is low and multiple lasers are active!

## ✅ Test to Confirm:

Add this debug print in your main update loop (or where you're handling lasers):

```
python

for laser in self.ship.lasers:
    print(f"Laser {id(laser)} at y={laser.rect.y}")
```

Also log alien hits like you did:

```
python

print(f"Laser {id(laser)} hit alien {id(alien)} at ({alien.rect.x}, {alien.rect.y})")
```

Then watch for this pattern:

- Two or more **laser IDs** very close in  $y$  value
- Each hitting a vertically adjacent alien

## 🔒 Optional Fix (if you want to allow only one laser on screen):

You could change the firing method to limit it:

```
python
```

```
if len(self.lasers) == 0:
    self.fire_laser()
```

Or add a time delay between shots to prevent "laser stacking."

But if you're okay with rapid fire, then... mystery solved: it's just really good aim with multiple shots 😎

Want help visualizing this by drawing rectangles or outlines of laser positions vs aliens? Or tweaking the laser cooldown?



There are two lasers, but I don't think it's rapid fire.

Laser 2097443798528 at y=745

Laser 2097443765968 at y=745

In my ship method, I created a method explicitly for single fire.  
And I updated the controls methods in events to reflect that.

```
import pygame as pg
import sys
from vector import Vector
```

```
class Event:
```

```
    di = {pg.K_RIGHT: Vector(1, 0), pg.K_LEFT: Vector(-1, 0),
          pg.K_UP: Vector(0, -1), pg.K_DOWN: Vector(0, 1),
          pg.K_d: Vector(1, 0), pg.K_a: Vector(-1, 0),
          pg.K_w: Vector(0, -1), pg.K_s: Vector(0, 1)}
```

```
    def __init__(self, ai_game):
        self.ai_game = ai_game
        self.settings = ai_game.settings
        self.stats = ai_game.stats
        self.sb = ai_game.sb
        self.game_active = ai_game.game_active
        self.ship = ai_game.ship
        self.play_button = ai_game.play_button
```

```
    def check_events(self):
        """Respond to keypresses and mouse events."""
        for event in pg.event.get():
            if (event.type == pg.QUIT) or (event.type == pg.KEYDOWN and (event.key == pg.K_q or
event.key == pg.K_ESCAPE)):
                sys.exit()
            return True # finished is True
            elif event.type == pg.MOUSEBUTTONDOWN:
                mouse_pos = pg.mouse.get_pos()
                self._check_play_button(mouse_pos)
```



```

        elif event.type == pg.MOUSEMOTION:
            mouse_pos = pg.mouse.get_pos()
            self.play_button.set_highlight(mouse_pos)
        else:
            self.control_ship()

def control_ship(self):
    self.ship.v = Vector(0, 0)
    keys = pg.key.get_pressed()

    # Movement checks
    if keys[pg.K_UP] or keys[pg.K_w]:
        self.ship.v += self.settings.ship_speed * Vector(0, -1)
    if keys[pg.K_DOWN] or keys[pg.K_s]:
        self.ship.v += self.settings.ship_speed * Vector(0, 1)
    if keys[pg.K_LEFT] or keys[pg.K_a]:
        self.ship.v += self.settings.ship_speed * Vector(-1, 0)
    if keys[pg.K_RIGHT] or keys[pg.K_d]:
        self.ship.v += self.settings.ship_speed * Vector(1, 0)

    # Firing check
    if keys[pg.K_SPACE]:
        # self.ship.open_fire()
        self.ship.single_fire()
    else:
        self.ship.cease_fire()

def _check_play_button(self, mouse_pos):
    """Start a new game when the player clicks Play."""
    button_clicked = self.play_button.rect.collidepoint(mouse_pos)
    if button_clicked and not self.ai_game.game_active:
        self.settings.initialize_dynamic_settings()
        self.ai_game.reset_game()

# def _check_keydown_events(self, event):
#     key = event.key
#     if key in Event.di.keys():
#         self.ship.v += self.settings.ship_speed * Event.di[key]
#     elif event.key == pg.K_SPACE:
#         self.ship.open_fire()
#     elif event.type == pg.KEYUP:
#         if event.key in Event.di.keys():
#             self.ship.v = Vector()
#         elif event.key == pg.K_SPACE:
#             self.ship.cease_fire()

# def _check_keyup_events(self, event):
#     if event.key in Event.di.keys():
#         self.ship.v = Vector()
#     elif event.key == pg.K_SPACE:
#         self.ship.cease_fire()

```

```

# System exit and start checks
# if keys[pg.K_ESCAPE]:
#     sys.exit()
# for event in pg.event.get():
#     if event.type == pg.QUIT:
#         sys.exit()
#     elif event.type == pg.MOUSEBUTTONDOWN:
#         mouse_pos = pg.mouse.get_pos()
#         self.check_play_button(mouse_pos)
#     elif event.type == pg.MOUSEMOTION:
#         mouse_pos = pg.mouse.get_pos()
#         self.button.set_highlight(mouse_pos)

```

It looks like you're on the right track, especially with the `single_fire()` method for ensuring one laser shot at a time. Given that you aren't experiencing rapid fire, the two lasers could still be generated because of how they're created or how the `single_fire()` method is handling the logic.

Here are some things to check or adjust in your code to ensure only one laser is fired per press:

### 1. Check `single_fire()` Method:

You might want to ensure that the laser isn't being created twice per space press due to multiple calls or unintended behavior. Verify that `single_fire()` is doing this:

```

python

def single_fire(self):
    if not self.lasers: # or another condition to check there's no active laser
        self.fire_laser()

```

### 2. Ensure Laser Creation is Exclusive:

Ensure `self.lasers` only contains one laser object by restricting its creation during an active fire. Check that `open_fire()` (or the equivalent method you're calling) only creates a laser if there's not already one active. You could also implement a timer or a cooldown to prevent firing again too quickly:

```

python

# In Ship class
def single_fire(self):
    if not self.lasers: # or use cooldown time if needed
        self.fire_laser()

```

This ensures that multiple lasers are not being created during a single `SPACE` press.

### 3. Spacebar Debounce Logic:

Sometimes, a single key press can result in multiple `KEYDOWN` events due to how the operating system processes key events. Try adding some debounce logic to prevent multiple rapid firings:

python

```
# inside control_ship
if keys[pg.K_SPACE] and not self.ship.is_firing:
    self.ship.single_fire()
    self.ship.is_firing = True # Prevents multiple firings until key is released
elif not keys[pg.K_SPACE]:
    self.ship.is_firing = False # Resets when the key is released
```

Make sure to define `self.is_firing` in the ship class to track whether the ship is in the middle of a firing action.

#### 4. Investigate Laser Object Handling:

It's possible that each time the **SPACE** bar is pressed, a new laser object is being created while the old one isn't removed yet. Double-check your laser removal logic to ensure the previous laser is properly removed or doesn't persist once the shot is fired.

#### 5. Double Check Event Handling:

It looks like you're correctly handling the space bar to trigger `single_fire`. Make sure there's no overlapping logic between `control_ship` and other event handlers. Your `single_fire` should ideally only execute once per key press.

---

Let me know if this solves the issue or if we need to dig deeper into any specific part!



The spacebar processing was literally the issue.

Awesome! I'm glad to hear that the spacebar issue is resolved. Now, with the single laser fire working, you should be able to avoid firing multiple lasers accidentally. Let me know if anything else comes up!



Thank you so much! It's both heart breaking and a relief to know that my original logic was sound and that the problem lied elsewhere.

You're very welcome! I totally get that feeling—it can be a mix of relief and frustration when you find out the issue was something simple after all the work you've put in. But hey, it's a great learning moment, and now your game is working as expected! Keep up the awesome work, and feel free to reach out if you run into anything else. You've got this! 😊



Alright, take care for now. Thank you for stealing all the information online so you can help me.

Haha, no worries! I'm here to help however I can, and I'm glad we could get things sorted out. Take care, and don't hesitate to reach out whenever you need more assistance. Have a great time with your game!

