

# Alien Invasion Row Setup



I am working on a project converting alien invasion to space invaders. I want to rework a portion of this code such that I specify the number of columns and use that to determine how many aliens are spawned in each row.

Since you want to specify the number of columns and use that to determine how many aliens spawn per row, you'll need to adjust your alien placement logic.

## Approach:

1. Set a fixed number of **columns**.
2. Use that number to determine the horizontal spacing.
3. Calculate how many **rows** fit based on available screen space.

Would you like me to adjust your current alien placement code, or would you like a fresh function to implement this? If you share your current fleet placement logic, I can refine it accordingly. 🚀



```
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):
    # 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, True, False)

    if collisions:
        for aliens in collisions.values():
            self.stats.score += self.settings.alien_points * len(aliens)
            for alien in aliens:
                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()
```

Thanks for sharing your `Fleet` class! To modify it so that the number of **columns** is specified and used to determine how many aliens appear per row, we need to adjust the `create_row` and `create_fleet` methods.

## Key Adjustments

1. **Specify the number of columns** (e.g., `self.num_columns = desired_value`).
2. **Calculate column spacing** based on screen width.
3. **Determine the number of rows** based on available height.

## Updated Code (with Column Control)

```
python

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

    # Set desired number of columns
    num_columns = 10 # Adjust as needed

    # 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 based on screen width
    total_width = self.settings.scr_width
    available_space_x = total_width - (self.spacing * tile_width)
    column_spacing = available_space_x / num_columns

    # Determine how many rows fit on the screen
    available_space_y = self.settings.scr_height - (self.spacing * 6 * tile_height)
    num_rows = int(available_space_y // (self.spacing * tile_height))

    # Create the fleet using specified columns and calculated rows
    for row in range(num_rows):
        y = (row + 1) * self.spacing * tile_height
        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."""
    for col in range(num_columns):
        x = (col + 1) * column_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)
```

## How It Works

- The **number of columns** is fixed (`num_columns`).
- The **column spacing** is determined dynamically based on screen width.
- The **number of rows** is determined dynamically based on screen height.
- `create_row` now takes `num_columns` and `column_spacing` as arguments.

## Benefits

- ✓ Keeps alien spacing even.
- ✓ Ensures the fleet fits dynamically within the screen.
- ✓ Maintains fine control over the number of columns.

Would you like to tweak the spacing further or make any other refinements? 🚀



For the spacing between columns and rows, I actually just want it based on the tile width and height.

Got it! If you want the spacing between columns and rows to be based solely on the **tile width and height**, we can simplify the calculations.

## Key Adjustments:

1. **Spacing between columns** = `tile_width * self.spacing`
2. **Spacing between rows** = `tile_height * self.spacing`
3. **Number of rows and columns are controlled purely by screen size**

## Updated Code:

```
python

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
    num_columns = 10 # Adjust as needed

    # 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

# Determine how many rows fit on the screen
num_rows = (self.settings.scr_height - (self.spacing * 6 * tile_height)) // (tile_height *
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)
```

## How It Works

- ✓ **Column spacing** is just `tile_width * spacing`.
- ✓ **Row spacing** is just `tile_height * spacing`.
- ✓ **Rows are dynamically determined** based on screen height.
- ✓ **Columns are fixed** to a given number (e.g., 10).

Would you like to make the number of columns dynamic as well based on screen width? 🚀