

Condense Sprite Scaling Loop



I want to condense this code so I only have to use 1 for loop for all sprites. I will create `self.ship_sprites` to hold `ship_images`, `ship_damage`, and `ship_boom`.

```
# Ship Scales
self.true_scale = self.settings.fighter_true_scale
self.ship_images = []
self.ship_boom = []
self.ship_damage = []

# Applying Image Scale
for ship_image in Ship.ship_images:
    self.ship_images.append(pg.transform.scale_by(ship_image, self.true_scale))

for ship_damage in Ship.ship_damage:
    self.ship_damage.append(pg.transform.scale_by(ship_damage, self.true_scale))

for ship_boom in Ship.ship_boom:
    self.ship_boom.append(pg.transform.scale_by(ship_boom, self.true_scale))
```

You can definitely condense that into a single loop by grouping the sprite lists together using `zip` and storing the results in a new `self.ship_sprites` dictionary or list of lists, depending on your preferred structure.

Here's a clean and compact version using a dictionary for clarity:

```
python

# Ship Scales
self.true_scale = self.settings.fighter_true_scale
self.ship_sprites = {
    "images": [],
    "damage": [],
    "boom": []
}

for src_list, key in zip(
    [Ship.ship_images, Ship.ship_damage, Ship.ship_boom],
    ["images", "damage", "boom"]
):
    self.ship_sprites[key] = [pg.transform.scale_by(img, self.true_scale) for img in src_list]
```

Now:

- `self.ship_sprites["images"]` holds the scaled ship images,
- `self.ship_sprites["damage"]` holds the scaled damage sprites,
- `self.ship_sprites["boom"]` holds the scaled explosion sprites.

If you still need `self.ship_images`, `self.ship_damage`, etc., for compatibility, you can optionally assign them afterward:

python

```
self.ship_images = self.ship_sprites["images"]
self.ship_damage = self.ship_sprites["damage"]
self.ship_boom = self.ship_sprites["boom"]
```

Let me know if you prefer a list-based structure instead!



What would a list-based structure look like?

A list-based structure could look like this: a list containing the three sprite lists in a fixed order — for example, `[images, damage, boom]`. This is more compact but slightly less readable unless you document or always remember the order.

Here's how you'd implement it:

python

```
# Ship Scales
self.true_scale = self.settings.fighter_true_scale
self.ship_sprites = []

for src_list in [Ship.ship_images, Ship.ship_damage, Ship.ship_boom]:
    scaled_list = [pg.transform.scale_by(img, self.true_scale) for img in src_list]
    self.ship_sprites.append(scaled_list)
```

Accessing the elements:

If you go this route, it's helpful to assign indices for clarity:

python

```
SHIP_IMAGES, SHIP_DAMAGE, SHIP_BOOM = 0, 1, 2
```

Then you can use:

python

```
self.ship_sprites[SHIP_IMAGES] # Scaled images
self.ship_sprites[SHIP_DAMAGE] # Scaled damage
self.ship_sprites[SHIP_BOOM]   # Scaled boom
```

Optional: assign to old variable names for compatibility

python

```
self.ship_images = self.ship_sprites[SHIP_IMAGES]
self.ship_damage = self.ship_sprites[SHIP_DAMAGE]
self.ship_boom = self.ship_sprites[SHIP_BOOM]
```

Let me know which structure feels cleaner for your use — dictionary if you want clarity and readability, or list if you're going for brevity and slightly faster access.



In the previous method, is `self.ship_sprites` a dict?

Yes — in the previous method (the one using `zip` and keys like `"images"`, `"damage"`, `"boom"`), `self.ship_sprites` is a **dictionary**.

Here's a quick breakdown again for clarity:

```
python

self.ship_sprites = {
    "images": [],
    "damage": [],
    "boom": []
}
```

Then this loop fills in the dictionary values:

```
python

for src_list, key in zip(
    [Ship.ship_images, Ship.ship_damage, Ship.ship_boom],
    ["images", "damage", "boom"]
):
    self.ship_sprites[key] = [pg.transform.scale_by(img, self.true_scale) for img in src_list]
```

So afterwards:

- `self.ship_sprites["images"]` → your scaled ship images
- `self.ship_sprites["damage"]` → your scaled damaged ship sprites
- `self.ship_sprites["boom"]` → your scaled explosion sprites

Let me know if you want to do the same pattern for enemies or other sprite groups too — we can make it easily reusable.



What does `src_list` stand for?

Great question! In this context, `src_list` stands for **"source list"** — it's each of the original sprite lists from the `Ship` class, before scaling.

Here's the loop again with a little annotation:

```
python

for src_list, key in zip(
    [Ship.ship_images, Ship.ship_damage, Ship.ship_boom], # these are the source lists
    ["images", "damage", "boom"] # the corresponding keys in the destination dictionary
):
```

```
) :
    # src_list is one of Ship.ship_images, Ship.ship_damage, or Ship.ship_boom
    self.ship_sprites[key] = [pg.transform.scale_by(img, self.true_scale) for img in src_list]
```

So on each iteration:

- `src_list` is one of the original unscaled lists: first `Ship.ship_images`, then `Ship.ship_damage`, then `Ship.ship_boom`.
- `key` is the corresponding string: `"images"`, `"damage"`, or `"boom"`.

The loop is just matching each source list to its name and applying scaling to all its images. It's like doing:

```
python

self.ship_sprites["images"] = [scaled images from Ship.ship_images]
self.ship_sprites["damage"] = [scaled images from Ship.ship_damage]
self.ship_sprites["boom"]   = [scaled images from Ship.ship_boom]
```

But all in one compact loop.



Let's change the approach for a moment. I am loading a bunch of information into a dict called `self.info`. Info has the source sprites. Ex: `self.info["sprites_base"]`, `self.info["sprites_boom"]`, etc. These will never change once loaded in.

Say I want to initialize the sprites as they are to `self.ship_sprites`, then use a `self.adjust_scale` method to adjust these sprites at will using the sprites in `self.info` as reference for scaling/transformation. How would I do that?

```
# Info
self.info = Ship.fighter_type[type % len(Ship.fighter_type)]

self.sprites = {
    "base": self.info["sprites_base"],
    "damage": self.info["sprites_damage"],
    "boom": self.info["sprites_boom"]
}
```

Ah, perfect — since you're treating `self.info` as a **static source of truth**, and you want `self.ship_sprites` to be a **mutable copy** that you can scale or otherwise transform based on the originals, here's a clean way to structure that.

1. Initialize `self.ship_sprites` from `self.info` (unscaled):

You're already doing this part right:

```
python

self.info = Ship.fighter_type[type % len(Ship.fighter_type)]
```

```
self.ship_sprites = {
    "base": self.info["sprites_base"][:],
    "damage": self.info["sprites_damage"][:],
    "boom": self.info["sprites_boom"][:]
}
```

Note: The `[:]` makes a shallow copy so that `self.ship_sprites` doesn't reference the same list as `self.info`.

2. Define `self.adjust_scale()` method:

This method will update `self.ship_sprites` by scaling from `self.info` — leaving the originals untouched.

```
python

def adjust_scale(self, scale):
    self.true_scale = scale # Optional: store it for reference
    for key in self.ship_sprites:
        self.ship_sprites[key] = [
            pg.transform.scale_by(sprite, scale) for sprite in self.info[f"sprites_{key}"]
        ]
```

Example usage:

```
python

self.adjust_scale(self.settings.fighter_true_scale)
```

Now you can change scales dynamically at runtime without losing the originals.

Bonus — If you might apply other transformations later (like tinting, flipping, etc.), this architecture is great. You can always base everything off `self.info`, treating it as a read-only template.

Let me know if you'd like to also cache other transforms (e.g., rotated, flipped, etc.) the same way!



Thank you; I am good for now. :)

You're very welcome! 😊 Glad I could help — and it sounds like your structure is really shaping up nicely. Feel free to swing by anytime if you want to fine-tune anything or dive into new features. Happy coding, and have fun with your game! 🚀💜