How would you handle interpolation in a python game?

Asked 12 years, 4 months ago Modified 12 years, 3 months ago Viewed 2k times

5

I'm currently writing a very simple game using python and pygame. It has stuff that moves. And to make this stuff move smoothly I arranged the main game loop as said in Fix Your Timestep, with interpolation.

Here is how I handle interpolation now.

```
class Interpolator(object):
    """Handles interpolation"""
    def __init__(self):
        self.ship_prev = None
        self.ship = None

        self.stars_prev = []
        self.stars = []

        self.bullets_prev = {}
        self.bullets = {}

        self.alpha = 0.5

    def add_ship(self, ship):
        self.ship_prev = self.ship
        self.ship = ship

    def add_stars(self, stars):
        self.stars_prev = self.stars
        self.stars = stars[:]

    def add_bullets(self, bullets):
        self.bullets_prev = self.bullets
        self.bullets = bullets.copy()

    def add_enemies(self, enemies):
        self.enemies_prev = self.enemies
        self.enemies = enemies  # to be continued

    def lerp_ship(self):
        if self.ship_prev is None:
            return self.ship
        return lerp_xy(self.ship_prev, self.ship, self.alpha)

    def lerp_stars(self):
        if len(self.stars_prev) == 0:
            return self.stars
        return (lerp_xy(s1, s2, self.alpha) for s1, s2 in izip(self.stars_prev,
self.stars))

    def lerp_bullets(self):
        keys = list(set(self.bullets_prev.keys() + self.bullets.keys()))
        for k in keys:
            # interpolate as usual
            if k in self.bullets_prev and k in self.bullets:
                yield lerp_xy(self.bullets_prev[k], self.bullets[k],
self.alpha)
            # bullet is dead
            elif k in self.bullets_prev:
                pass
            # bullet just added
            elif k in self.bullets:
                yield self.bullets[k]
```

The lerp_xy() function and data types

```
def lerp_xy(o1, o2, alpha, threshold=100):
    """Expects namedtuples with x and y parameters."""
    if sqrt((o1.x - o2.x) ** 2 + (o1.y - o2.y) ** 2) > 100:
        return o2
    return o1._replace(x=lerp(o1.x, o2.x, alpha), y=lerp(o1.y, o2.y, alpha))

Ship = namedtuple('Ship', 'x, y')
Star = namedtuple('Star', 'x, y, r')
Bullet = namedtuple('Bullet', 'x, y')
```

The data types are of course temporary, but I still expect they will have the x and y attributes in the future. **update:** lerper.alpha is updated each frame.

Ship is added as a single object - it's the player ship. Stars are added as a list. Bullets are added as a dict {id, Bullet}, since bullets are added and removed all the time, I have to track which bullet is which, interpolate if both are present and do something if it just has been added or deleted.

Anyway this code right here is crap. It grew as I added features, and now I want to rewrite it to be more generic, so it can continue growing and not become an unpythonic stinky pile of poo.

Now I'm still pretty new to Python, though I feel pretty comfortable with list comprehensions, generators and coroutines already.

What I have the least experience with is the OO side of Python, and designing an architecture of something bigger than a 10-line hacky disposable script.

The question is not a question as in something I don't know and can't do anything about it. I'm sure I'm capable of rewriting this pretty simple code that will work in some way close to what I want.

What I do want to know, is the way experienced Python programmers will solve this simple problem in a pythonic way, so I (and of course others) could learn what is considered an elegant way of handling such a case among Python developers.

So, what I approximately want to achieve, in pseudocode:

```
lerper = Interpolator()
# game loop
while(1):
    # physics
    # physics done
    lerper.add(ship)
    lerper.add(stars)
    lerper.add(bullets)
    lerper.add(enemies) # you got the idea

    # rendering
    draw_ship(lerper.lerp('Ship'))
    # or
    draw_ship(lerper.lerp_ship())
```

However don't let that pseudocode stop you if you have a better solution in mind =)

So. Make all game objects as separate/inherited class? Force them all to have id? Add them all as list/dict `lerper.add([ship])`? Make a special container class inheriting from dict/whatever? What do you consider an elegant, pythonic way of solving this? How would you do it?

python pygame

Share

Improve this question

Follow

edited Aug 1, 2012 at 19:06

asked Aug 1, 2012 at 18:13

Mikka

2,133 ● 2 ● 16 ● 12

3 Answers

Sorted by: Highest score (default)

2

This may not be what you are looking for, but it will hopefully nudge you in a helpful direction in trying to write a game. The following recipes written for Python 3.x provide an example of writing working modules.

- vector (provides a class for working with 2D coordinates and does more than `complex` numbers)
- processing (provides an extensible framework for creating an animation or making a simple game)
- boids (demonstrates how to create a structured animation that runs independently from frame rate)

You might look at the code provided above and use it as an inspiration for writing your framework or structuring your code so it becomes reusable. The referenced project was inspired by Processing.org.

Share Improve this answer Follow

answered Aug 1, 2012 at 19:13
Noctis Skytower
22k ● 16 ● 84 ● 121

1

I wrote a half-finished Breakout clone that has similar "objects moving around" code as yours, so I'll share how I did it.

Anything with a position and velocity is instantiated from the Projectile class. Non-moving objects can be projectiles too. A projectile is responsible for updating its own position when somebody calls `tick` on it.

```
class Projectile:
    def __init__(self):
        self.x = 0
        self.y = 0
        self.vel_x = 0
        self.vel_y = 0
    def tick(self, dt):
        self.x += dt * self.vel_x
        self.y += dt * self.vel_y
```

Later, you may want to replace `x` and `y` with something like a axis-aligned bounding box, so you can do collision detection between projectiles.

All projectiles that interact with one another live in a Layer, which is responsible for `tick`ing each projectile.

```
class Layer:
    def __init__(self):
        self.projectiles = []
    def addProjectile(self, p):
        self.projectiles.add(p)
    def tick(self, dt):
        for p in self.projectiles:
            p.tick(dt)
        #possible future expansion: put collision detection here
```

Set up is just a matter of creating game objects with the properties you want, and adding them to a layer.

```
#setup
l = Layer()

ship = Projectile()
```

```
#not shown: set initial position and velocity of ship
l.addProjectile(ship)

for i in range(numStars):
    star = Projectile()
    #not shown: set initial position and velocity of stars
    l.addProjectile(star)

#not shown: add bullets and enemies to l, the same way stars were

#game loop
while True:
    #get dt somehow
    dt = .42
    l.tick(dt)
    for projectile in l.projectiles:
        draw(l)
```

Drawing is where your program and mine diverge - everything in breakout is a rectangle, so every game object can be drawn in the same way. But a space invader doesn't look like a star, and a bullet doesn't look like a space ship, so they would all need unique code. At this point, you should consider making Ship, Star, Bullet, Enemy, etc, which are subclasses of Projectile. Then each one can specify their own appearance. They could also have individual behaviors beyond moving in straight lines - ex. Ships respond to keypresses, Enemies accelerate towards ships, etc.

Share Improve this answer Follow

answered Aug 1, 2012 at 19:04

Kevin
76.1k ● 13 ● 138 ● 167

1 Did you read the whole question? I don't see anything related to interpolation or the problem I presented in your answer. The only "idea" I see in your answer is "make each game object handle its own interpolation", but you don't even mention it O.o – Mikka Aug 1, 2012 at 19:11

Perhaps I misunderstood your question. I took it to mean, "How do I revise my game so that it is more generic, and can be easily expanded?". If your question was actually "How do I revise my interpolation algorithm to be more generic?", then you have already guessed my opinion: make each game object handle its own interpolation. (preferably only once, in the Projectile base class) – Kevin Aug 1, 2012 at 19:19

1

Here is how I ended up handling the interpolation:

```
class Thing(object):
    """Generic game object with interpolation"""
    def __init__(self, x=0, y=0):
        self._x = self.x = x
        self._y = self.y = y

    def imprint(self):
        """call before changing x and y"""
        self._x = self.x
        self._y = self.y

    def __iter__(self):
        """handy to unpack like a tuple"""
        yield self.x
        yield self.y

Ship = Thing
Bullet = Thing


class Star(Thing):
    """docstring for Star"""
    def __init__(self, x, y, r):
        super(Star, self).__init__(x, y)
        self.r = r

    def __iter__(self):
        yield self.x
        yield self.y
        yield self.r


def lerp_things(things, alpha, threshold=100):
    """Expects iterables of Things"""
    for t in things:
        if sqrt((t._x - t.x) ** 2 + (t._y - t.y) ** 2) > threshold:
            yield (t.x, t.y)
        else:
            yield (lerp(t._x, t.x, alpha), lerp(t._y, t.y, alpha))
```

Share

Improve this answer

Follow

edited Sep 24, 2012 at 16:50

Gareth Rees

65.8k ● 10 ● 137 ● 165

answered Aug 6, 2012 at 20:08

Mikka

2,133 ● 2 ● 16 ● 12