```python
import random
import pygame

WIDTH, HEIGHT = 1200, 700
BACKGROUND = (10, 10, 18)


def note_to_x(note, width):
    min_note, max_note = 21, 108
    note = max(min_note, min(max_note, note))
    return int((note - min_note) / (max_note - min_note) * width)


def note_to_color(note, velocity):
    r = (note * 5) % 256
    g = (note * 9) % 256
    b = (note * 13) % 256
    boost = int((velocity / 127) * 80)

    return (
        min(255, r + boost),
        min(255, g + boost),
        min(255, b + boost),
    )


class Droplet:
    def __init__(self, x, y, radius, color, growth, alpha_decay):
        self.x = x
        self.y = y
        self.radius = radius
        self.color = color
        self.growth = growth
        self.alpha = 255
        self.alpha_decay = alpha_decay

    def update(self):
        self.radius += self.growth
        self.alpha -= self.alpha_decay

    def is_alive(self):
        return self.alpha > 0
```

```
    def draw(self, surface):
        if self.alpha <= 0:
            return

        size = int(self.radius * 2 + 20)
        temp = pygame.Surface((size, size), pygame.SRCALPHA)
        draw_color = (*self.color, max(0, int(self.alpha)))
        pygame.draw.circle(temp, draw_color, (size // 2, size // 2), int(self.radius))
        surface.blit(temp, (self.x - size // 2, self.y - size // 2))


class WaterfallBar:
    def __init__(self, x, width, height, speed, color):
        self.x = x
        self.y = -height
        self.width = width
        self.height = height
        self.speed = speed
        self.color = color
        self.alpha = 255

    def update(self):
        self.y += self.speed
        if self.y > HEIGHT * 0.6:
            self.alpha -= 6

    def is_alive(self):
        return self.alpha > 0 and self.y < HEIGHT + self.height

    def draw(self, surface):
        temp = pygame.Surface((self.width, self.height), pygame.SRCALPHA)
        draw_color = (*self.color, max(0, int(self.alpha)))
        pygame.draw.rect(temp, draw_color, (0, 0, self.width, self.height),
border_radius=8)
        surface.blit(temp, (self.x, self.y))


class Spark:
    def __init__(self, x, y, dx, dy, size, color, life):
        self.x = x
        self.y = y
```

```
        self.dx = dx
        self.dy = dy
        self.size = size
        self.color = color
        self.life = life
        self.max_life = life

    def update(self):
        self.x += self.dx
        self.y += self.dy
        self.dy += 0.08
        self.life -= 1

    def is_alive(self):
        return self.life > 0

    def draw(self, surface):
        alpha = int(255 * (self.life / self.max_life))
        temp = pygame.Surface((self.size * 4, self.size * 4), pygame.SRCALPHA)
        draw_color = (*self.color, alpha)
        pygame.draw.circle(temp, draw_color, (self.size * 2, self.size * 2), self.size)
        surface.blit(temp, (self.x, self.y))


class BaseVisualizer:
    def __init__(self, width=WIDTH, height=HEIGHT):
        self.width = width
        self.height = height

    def add_note(self, note, velocity):
        pass

    def update(self):
        pass

    def draw(self, surface):
        surface.fill(BACKGROUND)


class DropletsVisualizer(BaseVisualizer):
    def __init__(self, width=WIDTH, height=HEIGHT):
        super().__init__(width, height)
```

```
        self.droplets = []

    def add_note(self, note, velocity):
        x = note_to_x(note, self.width)
        y = random.randint(self.height // 3, self.height - self.height // 4)
        radius = 20 + int((velocity / 127) * 60)
        color = note_to_color(note, velocity)
        growth = random.uniform(0.5, 1.5)
        alpha_decay = random.uniform(3, 6)
        self.droplets.append(Droplet(x, y, radius, color, growth, alpha_decay))

    def update(self):
        for d in self.droplets:
            d.update()
        self.droplets = [d for d in self.droplets if d.is_alive()]

    def draw(self, surface):
        surface.fill(BACKGROUND)
        for d in self.droplets:
            d.draw(surface)


class WaterfallVisualizer(BaseVisualizer):
    def __init__(self, width=WIDTH, height=HEIGHT):
        super().__init__(width, height)
        self.bars = []

    def add_note(self, note, velocity):
        x = note_to_x(note, self.width)
        bar_width = 14
        bar_height = 80 + int((velocity / 127) * 140)
        speed = 4 + (velocity / 127) * 6
        color = note_to_color(note, velocity)
        self.bars.append(WaterfallBar(x, bar_width, bar_height, speed, color))

    def update(self):
        for b in self.bars:
            b.update()
        self.bars = [b for b in self.bars if b.is_alive()]

    def draw(self, surface):
        surface.fill((5, 8, 18))
```

```
        for b in self.bars:
            b.draw(surface)


class SparksVisualizer(BaseVisualizer):
    def __init__(self, width=WIDTH, height=HEIGHT):
        super().__init__(width, height)
        self.sparks = []

    def add_note(self, note, velocity):
        x = note_to_x(note, self.width)
        y = self.height // 2
        color = note_to_color(note, velocity)

        num_sparks = 12 + velocity // 10
        for _ in range(num_sparks):
            dx = random.uniform(-4, 4)
            dy = random.uniform(-6, -1)
            size = random.randint(2, 5)
            life = random.randint(20, 40)
            self.sparks.append(Spark(x, y, dx, dy, size, color, life))

    def update(self):
        for s in self.sparks:
            s.update()
        self.sparks = [s for s in self.sparks if s.is_alive()]

    def draw(self, surface):
        surface.fill((12, 10, 10))
        for s in self.sparks:
            s.draw(surface)
```