Feature Implementation Instructions: Breath Kasina

Overview:
We're adding a new category called "Breath Kasinas" to our existing KASINA application (www.kasina.app). This new category will complement our current Color, Elemental, & Vajrayana Kasina sets, while being distinct among these other sets, in that it's feature we want to any Kasina once it's active. We will begin with the "Expand & Contract" kasina visualization, driven by real-time breath data from the Vernier Go Direct Respiration Belt. Later we'll add "Brighten & Darken" and "Changing Colors" as some other breath kasina effects that you could layer on, along with, or instead of, Expand & Contract.

Hardware Integration

- **Device:** Vernier Go Direct Respiration Belt
- **Communication:** Web Bluetooth API
- **Data Type:** Continuous numeric breath amplitude data
- **Data Update Rate:** ~20 Hz (updates every ~50 ms)

Integration Steps:

1. Connect via Web Bluetooth, using JavaScript in our Replit environment.
2. Identify BLE Service and Characteristic UUIDs (using nRF Connect or Vernier documentation).
3. Stream real-time breath amplitude data to the web app.

Visualization Details: Breath Kasina: Expand & Contract

- **Primary Visual Element:** A spherical orb. This effect applies to any of the other kasinas built into the app from the other sets.
- **Animation Behavior:**

    - Orb expands smoothly with inhalation and contracts gently with exhalation.
    - Orb's size directly corresponds to real-time breath amplitude (normalized).
- **Breathing Rate Adaptation:**
    - Orb visual intensity and maximum size dynamically adjust based on user's breathing rate.
    - As the breathing rate decreases, indicative of deeper meditation, the overall visual intensity and orb movements become subtler.
    - Specifically, a breathing rate of ~4 breaths per minute corresponds to near visual stillness, promoting deep concentration and calm.
- **Algorithm for Adaptive Visual Scaling:**

```
function updateOrbVisualization(rawBreathValue, currentBreathsPerMinute) {
  const normalizedAmplitude = (rawBreathValue - minVal) / (maxVal - minVal);

  // Breathing rate-based intensity scaling
  const intensityScale = Math.min(currentBreathsPerMinute / 12, 1); // Normal breath rate ~12/min
  const scaleFactor = 1 + (0.3 * normalizedAmplitude * intensityScale);

  orb.scale.setScalar(scaleFactor);

  // At 4 breaths per minute (or lower), intensityScale approaches zero, causing subtle, minimal visual motion.
}
```

User Experience (UX)

- **New Category Placement:** Listed alongside "Color Kasinas" and "Elemental Kasinas" in the main selection menu.
- **User Setup:**
    - Button labeled "Connect Breath Sensor" initiates Web Bluetooth connection.

- Short calibration phase (10–20 seconds) to establish personalized min/max breathing amplitude.

Technical Specifications for Replit Implementation

- Use Replit's built-in Node.js and frontend frameworks (Three.js or Canvas 2D).
- Web Bluetooth API for BLE integration: Chrome desktop and Android-compatible.
- Ensure graceful handling of connection errors and provide clear feedback to the user.

Testing & Validation Criteria

- Successfully streaming and visualizing real-time breath data.
- Smooth, responsive visual scaling without noticeable latency (below 100 ms).
- Proper adaptive intensity response verified through breathing rate changes (from normal ~12 breaths/min down to 4 breaths/min).

🧘‍♂️ Replit AI Agent Instructions: Build Breath Kasina Integration (from scratch)

🎯 Goal:

Build a Web Bluetooth-powered feature that connects to the Vernier Go Direct Respiration Belt, sends the correct activation command, and visualizes real-time breath data by animating a glowing orb (the “Breath Kasina”) based on user breathing.

✅ Feature Requirements

1. Bluetooth Connection Flow

- Prompt the user to connect to a BLE device with name starting with `"GDX-RB"`
- Connect to:
    - **Command Characteristic UUID:**
      `f4bf14a6-c7d5-4b6d-8aa8-df1a7c83adcb`
    - **Response Characteristic UUID:**
      `b41e6675-a329-40e0-aa01-44d2f444babe`

---

2. Send Activation Command

Once connected, send this activation command to the **command characteristic**:

```
const enableSensorCommand = new Uint8Array([
  0x58, 0x19, 0xFE, 0x3F, 0x1A, 0xA5, 0x4A, 0x06,
  0x49, 0x07, 0x48, 0x08, 0x47, 0x09, 0x46, 0x0A,
  0x45, 0x0B, 0x44, 0x0C, 0x43, 0x0D, 0x42, 0x0E, 0x41
]);
await commandCharacteristic.writeValue(enableSensorCommand);
console.log("✅ Sensor activation command sent");
```

---

3. Start Notifications from the Belt

Enable notifications on the **response characteristic**, and log every incoming data packet:

```
await responseCharacteristic.startNotifications();
responseCharacteristic.addEventListener("characteristicvaluechanged", (event) => {
  const raw = new Uint8Array(event.target.value.buffer);
  console.log("✅ Raw BLE data received:", raw);
  handleBreathData(raw);
});
```

---

4. Decode and Normalize the Breath Force (Placeholder)

In the `handleBreathData(raw)` function:

For now, use a placeholder decoder like:

```
 function handleBreathData(raw) {
  const force = raw[3] / 255; // TEMP: crude normalization
  updateOrb(force);
}
```

-
- **Later**, Vince will provide a more accurate decoding function based on actual breathing trials.

---

5. Visualize the Breath Orb ("Breath Kasina")

Create a simple animated orb in the UI:

- Use `HTMLCanvas`, `SVG`, or CSS
- Animate:
    - **Size or glow** based on normalized `force`
    - (Optional) Color or pulsing speed based on `respirationRate` (if extracted later)

```
function updateOrb(force) {
  const orb = document.getElementById("breath-orb");
  orb.style.transform = `scale(${0.8 + force * 0.6})`;
  orb.style.opacity = 0.5 + force * 0.5;
}
```

---

6. Fallbacks and Errors

- Handle common errors gracefully:
    - Device not found
    - BLE connection timeout

- No data received

Provide user feedback like:

console.warn("⚠️ No BLE data received. Is your belt already connected to another app?");

✅ Summary of Deliverables

Please implement the following in a clean, reusable, well-commented way:

- `connectToBreathBelt()` — initializes BLE pairing and writes activation command
- `handleBreathData(raw)` — decodes incoming sensor values
- `updateOrb(force)` — visualizes force in real time
- Clean HTML/CSS UI with a centered animated orb
- Console logs for raw BLE data and user feedback

Once this foundation is complete and tested, Vince will refine the `handleBreathData()` function to decode real breath force accurately from observed BLE packet formats.

Let me know when it's ready for decoding help. 🫁✨