```jsx
import React, { useEffect, useRef, useState } from "react";
import { fromUrl } from "geotiff";
import ScreenControls from "./ScreenControls";
import VirusControls, { DEFAULTS as VIRUS_DEFAULTS } from "./VirusControls";
import "../CSS/index.css";

// LiveSim: main component!
// - Takes the population geotiff file and turns it into a visual (dots are multiplied by population to get size).
// - Polls a binary RGB frame (sim_frame.bin) and paints population-scaled circles
// - Handles zoom
//   Has a virus panel (values are lifted into this component so they stay there when panels toggle and don't reset)
export default function LiveSim() {
  const canvasRef = useRef(null);

  const simBufRef = useRef(null);
  const popResRef = useRef(null);
  const maxPopRef = useRef(1);

  const rafIdRef = useRef(null);
  const pollIntervalRef = useRef(null);
  const pollAbortRef = useRef(null);

  const zoomRef = useRef(1);
  const offsetRef = useRef({ x: 0, y: 0 });

  const drawStepRef = useRef(2);
  const globalMultRef = useRef(1);
  const compressExpRef = useRef(0.7);
  const percentileCapRef = useRef(0.95);
  const minRadiusRef = useRef(0.1);
  const maxRadiusRef = useRef(0.9);

  const startedRef = useRef(false);
  const listenersAddedRef = useRef(false);

  const BASE_W = 1440;
  const BASE_H = 720;

  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);
  const [openPanel, setOpenPanel] = useState(null);

  const [controls, setControls] = useState({
    drawStep: 2,
    globalMultiplier: 1,
    compressExp: 0.7,
    percentileCap: 0.95,
```

```
      minRadius: 0.1,
      maxRadius: 0.9,
    });

    const [virusValues, setVirusValues] = useState(() => VIRUS_DEFAULTS);

    const togglePanel = (p) => {
      setOpenPanel((v) => (v === p ? null : p));
    };

    // Load & resample population
    // I fetch a GeoTIFF and use it in it the fixed BASE_W x BASE_H grid used for rendering.
    // This simplifies mapping between population values and canvas pixels so sizing remains predictable.
    useEffect(() => {
      let alive = true;

      const load = async () => {
        try {
          const tiff = await fromUrl("/population.tif");
          const img = await tiff.getImage();
          const w = img.getWidth();
          const h = img.getHeight();
          const ras = await img.readRasters({ interleave: true });

          // convert to single-band float array (assume first band is population)
          const spp = ras.length / (w * h);
          const raw = new Float32Array(w * h);
          for (let i = 0; i < w * h; i++) raw[i] = ras[i * spp] || 0;

          // simple nearest-neighbor downsample to our base sim grid
          const resized = new Float32Array(BASE_W * BASE_H);
          for (let y = 0; y < BASE_H; y++) {
            const sy = Math.floor((y / BASE_H) * h);
            for (let x = 0; x < BASE_W; x++) {
              const sx = Math.floor((x / BASE_W) * w);
              resized[y * BASE_W + x] = raw[sy * w + sx] || 0;
            }
          }

          popResRef.current = resized;

          // compute a percentile-based max for nicer radius scaling
          const sorted = Array.from(resized).sort((a, b) => a - b);
          const idx = Math.floor(sorted.length * percentileCapRef.current);
          maxPopRef.current = sorted[idx] || 1;

          if (alive) setLoading(false);
        } catch (e) {
          if (alive) {
```

```
            setError(e.message || String(e));
            setLoading(false);
          }
        }
      };

      load();

      return () => {
        alive = false;
      };
    }, []);

    // Main render loop
    // Reads latest binary RGB frame  and paints circles
    // - Uses exponent used to compress circles so that they get larger with increased population but don't get too too big
    useEffect(() => {

      const canvas = canvasRef.current;
      const ctx = canvas.getContext("2d");

      canvas.width = BASE_W;
      canvas.height = BASE_H;

      const draw = () => {
        const sim = simBufRef.current;
        const pop = popResRef.current;

        if (sim && pop) {
          ctx.clearRect(0, 0, BASE_W, BASE_H);

          // apply zoom and offset so zoom is anchored to user interactions
          const z = zoomRef.current;
          const off = offsetRef.current;
          ctx.setTransform(z, 0, 0, z, off.x, off.y);

          const step = drawStepRef.current || 1;
          const maxPop = maxPopRef.current;
          const minR = minRadiusRef.current * globalMultRef.current;
          const maxR = maxRadiusRef.current * globalMultRef.current;
          const exp = compressExpRef.current;

          // iterate over the downsampled grid and draw a colored circle where population exists
          for (let y = 0; y < BASE_H; y += step) {
            for (let x = 0; x < BASE_W; x += step) {
              const idx = y * BASE_W + x;
              const p = pop[idx];
              if (!p) continue;

              // normalize with a log scale then apply compression exponent
```

```
              let n = Math.log(p + 1) / Math.log(maxPop + 1);
              if (n < 0 || !isFinite(n)) n = 0;

              const r = minR + Math.pow(n, exp) * (maxR - minR);

              const o = idx * 3;
              ctx.fillStyle = `rgb(${sim[o]},${sim[o + 1]},${sim[o + 2]})`;
              ctx.beginPath();
              ctx.arc(x + 0.5, y + 0.5, r, 0, Math.PI * 2);
              ctx.fill();
            }
          }
        }

        rafIdRef.current = requestAnimationFrame(draw);
      };

      if (!startedRef.current) {
        startedRef.current = true;
        rafIdRef.current = requestAnimationFrame(draw);
      }

      return () => {
        if (rafIdRef.current) cancelAnimationFrame(rafIdRef.current);
        rafIdRef.current = null;
        startedRef.current = false;
      };
    }, []);

    // Poll the backend for the latest simulation frame (binary interleaved R
GB)
    // We want to change this to pull directly from receiver in the future in
stead of the sim_frame file, but we don't know if that's possible.
    useEffect(() => {
      const poll = async () => {
        if (pollAbortRef.current) pollAbortRef.current.abort();
        const ac = new AbortController();
        pollAbortRef.current = ac;

        try {
          const r = await fetch("/sim_frame.bin?cb=" + Date.now(), {
            signal: ac.signal,
          });
          if (!r.ok) return;
          const buf = await r.arrayBuffer();
          const u8 = new Uint8Array(buf);
          // basic validation: expect width * height * 3 bytes
          if (u8.length === BASE_W * BASE_H * 3) simBufRef.current = u8;
        } catch {}
        pollAbortRef.current = null;
      };

```

```
    poll();
    pollIntervalRef.current = setInterval(poll, 250);

    return () => {
      clearInterval(pollIntervalRef.current);
      pollIntervalRef.current = null;
      if (pollAbortRef.current) pollAbortRef.current.abort();
    };
  }, []);

  // Add wheel zoom handler (cursor-anchored zoom)
  // Zoom calculation keeps the point under the cursor stationary in canvas coordinates.
  useEffect(() => {
    const canvas = canvasRef.current;
    if (!canvas || listenersAddedRef.current) return;

    const onWheel = (e) => {
      e.preventDefault();
      const rect = canvas.getBoundingClientRect();
      const mx = e.clientX - rect.left;
      const my = e.clientY - rect.top;

      const prev = zoomRef.current;
      const next = Math.min(20, Math.max(1, prev * Math.exp(-e.deltaY * 0.001)));

      const s = next / prev;

      // adjust offset so zoom anchors on cursor position
      offsetRef.current.x = mx - (mx - offsetRef.current.x) * s;
      offsetRef.current.y = my - (my - offsetRef.current.y) * s;
      zoomRef.current = next;
    };

    canvas.addEventListener("wheel", onWheel, { passive: false });
    listenersAddedRef.current = true;

    return () => {
      canvas.removeEventListener("wheel", onWheel);
      listenersAddedRef.current = false;
    };
  }, []);

  // handleControlChange: central place to update React state. Refs are updated as well.
  const handleControlChange = (k, v) => {
    setControls((c) => ({ ...c, [k]: v }));

    // update the refs used in rendering to avoid re-wiring RAF loop
    if (k === "drawStep") drawStepRef.current = v;
    if (k === "globalMultiplier") globalMultRef.current = v;
    if (k === "compressExp") compressExpRef.current = v;
```

```
246       if (k === "minRadius") minRadiusRef.current = v;
247       if (k === "maxRadius") maxRadiusRef.current = v;
248
249       // percentile cap affects the computed `maxPop` used for normalization
250       if (k === "percentileCap") {
251         percentileCapRef.current = v;
252         if (popResRef.current) {
253           const a = Array.from(popResRef.current).sort((x, y) => x - y);
254           maxPopRef.current = a[Math.floor(a.length * v)] || 1;
255         }
256       }
257     };
258
259     // resetView_Controls: restore display-related parameters to sensible def
aults (will be fine tuned in the future)
260     // Note: we update both UI state and internal refs used by the renderer.
261     const resetView_Controls = () => {
262       zoomRef.current = 1;
263       offsetRef.current = { x: 0, y: 0 };
264
265       const d = {
266         drawStep: 2,
267         globalMultiplier: 1,
268         compressExp: 0.7,
269         percentileCap: 0.95,
270         minRadius: 0.1,
271         maxRadius: 0.9,
272       };
273
274       setControls(d);
275       drawStepRef.current = d.drawStep;
276       globalMultRef.current = d.globalMultiplier;
277       compressExpRef.current = d.compressExp;
278       percentileCapRef.current = d.percentileCap;
279       minRadiusRef.current = d.minRadius;
280       maxRadiusRef.current = d.maxRadius;
281     };
282
283     // virus handlers: simple setters lifted up so values persist across pane
l toggles
284     const handleVirusChange = (k, v) => {
285       setVirusValues((p) => ({ ...p, [k]: v }));
286     };
287
288     const resetVirusControls = () => {
289       // restore defaults defined in `VirusControls` so both components agree
290       setVirusValues(VIRUS_DEFAULTS);
291     };
292
293     return (
294       <div className="live-sim-container">
295         <div className="live-sim-status">
```

```
296           {loading && <p className="live-sim-loading">Loading population dat
a…</p>}
297           {error && <p className="live-sim-error">Error: {error}</p>}
298         </div>
299
300         <div className="live-sim-row">
301           <div className={`live-sim-controls-wrapper left ${openPanel ? "expa
nded" : "collapsed"}`}>
302             <div className="panel-header stacked">
303               <button
304                 className="panel-toggle"
305                 aria-controls="virus-panel"
306                 aria-expanded={openPanel === "virus"}
307                 onClick={() => togglePanel("virus")}
308               >
309                 Virus
310               </button>
311
312               <button
313                 className="panel-toggle"
314                 aria-controls="screen-panel"
315                 aria-expanded={openPanel === "screen"}
316                 onClick={() => togglePanel("screen")}
317               >
318                 Screen
319               </button>
320             </div>
321
322             {openPanel === "virus" && (
323               <div id="virus-panel">
324                 <VirusControls
325                   values={virusValues}
326                   onChange={handleVirusChange}
327                   onVirusReset={resetVirusControls}
328                 />
329               </div>
330             )}
331
332             {openPanel === "screen" && (
333               <div id="screen-panel">
334                 <ScreenControls
335                   values={controls}
336                   onChange={handleControlChange}
337                   onScreenReset={resetView_Controls}
338                 />
339               </div>
340             )}
341           </div>
342
343           <div className="live-sim-canvas-wrapper center">
344             <canvas
345               ref={canvasRef}
```

```
              className={`live-sim-canvas ${loading || error ? "hidden" : ""}
`}
            />
          </div>
        </div>
      </div>
    );
  }
```