```
import * as THREE from 'https://cdn.jsdelivr.net/npm/three@0.132.2/build/three.module.js';
import { OrbitControls } from
'https://cdn.jsdelivr.net/npm/three@0.132.2/examples/jsm/controls/OrbitControls.js'; // Scene
setup const container = document.getElementById('brain-container'); const scene = new
THREE.Scene(); scene.background = new THREE.Color(0xf0f0f0); // Camera setup const
camera = new THREE.PerspectiveCamera(75, window.innerWidth / window.innerHeight, 0.1,
1000); camera.position.z = 5; // Renderer setup const renderer = new
THREE.WebGLRenderer({ antialias: true }); renderer.setSize(window.innerWidth,
window.innerHeight); container.appendChild(renderer.domElement); // Controls const controls =
new OrbitControls(camera, renderer.domElement); controls.enableDamping = true;
controls.dampingFactor = 0.25; // Lighting const ambientLight = new
THREE.AmbientLight(0x404040, 1.5); scene.add(ambientLight); const directionalLight = new
THREE.DirectionalLight(0xffffff, 1); directionalLight.position.set(1, 1, 1);
scene.add(directionalLight); // Brain regions with colors and positions const regions = [ { name:
'Frontal/Prefrontal Cortex', color: 0x5599ff, position: new THREE.Vector3(0, 1.2, 0.6), scale: new
THREE.Vector3(1.2, 0.7, 0.8) }, { name: 'Parietal Cortex', color: 0x55dd55, position: new
THREE.Vector3(0, 1.2, -0.4), scale: new THREE.Vector3(1.2, 0.7, 0.8) }, { name: 'Left Temporal
Lobe', color: 0xdd55dd, position: new THREE.Vector3(-1.1, 0.3, 0.2), scale: new
THREE.Vector3(0.6, 0.7, 0.8) }, { name: 'Limbic System', color: 0xff9955, position: new
THREE.Vector3(0, 0, 0), scale: new THREE.Vector3(0.8, 0.7, 0.7) }, { name: 'Hippocampus',
color: 0xffff00, position: new THREE.Vector3(0.4, -0.2, 0.2), scale: new THREE.Vector3(0.3, 0.2,
0.5) }, { name: 'Basal Ganglia', color: 0xff5555, position: new THREE.Vector3(0.4, 0.2, 0), scale:
new THREE.Vector3(0.5, 0.5, 0.5) }, { name: 'Striatum', color: 0xff88bb, position: new
THREE.Vector3(0.6, 0.2, 0.2), scale: new THREE.Vector3(0.3, 0.3, 0.3) }, { name: 'Cerebellum',
color: 0xcc44cc, position: new THREE.Vector3(0, -1, -0.2), scale: new THREE.Vector3(1, 0.6,
0.7) }, { name: 'Dendritic Damage Areas', color: 0xff0000, position: new THREE.Vector3(-0.6,
0.5, 0.4), scale: new THREE.Vector3(0.2, 0.2, 0.2), opacity: 0.7 }, { name: 'Receptor Damage
Areas', color: 0xffaa00, position: new THREE.Vector3(0.6, 0.5, 0.6), scale: new
THREE.Vector3(0.2, 0.2, 0.2), opacity: 0.7 } ]; // Brain base (ellipsoid) const brainGeometry =
new THREE.SphereGeometry(1.5, 32, 32); const brainMaterial = new
THREE.MeshLambertMaterial({ color: 0xeeeeee, transparent: true, opacity: 0.2, wireframe: true
}); const brain = new THREE.Mesh(brainGeometry, brainMaterial); brain.scale.set(1, 0.8, 1.2);
scene.add(brain); // Create region meshes const regionMeshes = []; regions.forEach(region => {
const geometry = new THREE.SphereGeometry(1, 32, 16); const material = new
THREE.MeshLambertMaterial({ color: region.color, transparent: true, opacity: region.opacity ||
0.85 }); const mesh = new THREE.Mesh(geometry, material);
mesh.position.copy(region.position); mesh.scale.copy(region.scale); mesh.userData = { name:
region.name }; scene.add(mesh); regionMeshes.push(mesh); }); // Create labels const
textLabels = []; regions.forEach(region => { const labelDiv = document.createElement('div');
labelDiv.className = 'label'; labelDiv.textContent = region.name; labelDiv.style.position =
'absolute'; container.appendChild(labelDiv); textLabels.push({ element: labelDiv, position: new
THREE.Vector3().copy(region.position) }); }); // Raycaster for interaction const raycaster = new
THREE.Raycaster(); const mouse = new THREE.Vector2(); let selectedRegion = null; const
regionInfoPanel = document.getElementById('region-info'); const regionNameElement =
document.getElementById('region-name'); const onMouseMove = (event) => { // Calculate
mouse position in normalized device coordinates mouse.x = (event.clientX / window.innerWidth)
* 2 - 1; mouse.y = -(event.clientY / window.innerHeight) * 2 + 1; // Update the picking ray with the
camera and mouse position raycaster.setFromCamera(mouse, camera); // Calculate objects
intersecting the picking ray const intersects = raycaster.intersectObjects(regionMeshes); if
(intersects.length > 0) { const region = intersects[0].object; selectedRegion =
region.userData.name; document.body.style.cursor = 'pointer'; regionInfoPanel.style.display =
'block'; regionNameElement.textContent = selectedRegion; } else { selectedRegion = null;
document.body.style.cursor = 'auto'; regionInfoPanel.style.display = 'none'; } };
window.addEventListener('mousemove', onMouseMove); // Handle window resize const
handleResize = () => { camera.aspect = window.innerWidth / window.innerHeight;
camera.updateProjectionMatrix(); renderer.setSize(window.innerWidth, window.innerHeight); };
```

```
window.addEventListener('resize', handleResize); // Animation loop const animate = () => {
requestAnimationFrame(animate); // Update label positions textLabels.forEach(label => { const
pos = label.position.clone(); pos.project(camera); const x = (pos.x * 0.5 + 0.5) *
window.innerWidth; const y = (-pos.y * 0.5 + 0.5) * window.innerHeight;
label.element.style.transform = `translate(-50%, -50%) translate(${x}px, ${y}px)`; });
controls.update(); renderer.render(scene, camera); }; animate();
```