**EXPERIMENT NO 2.3**

## Aim

To design and implement an Interactive SVG Drawing Tool using HTML, CSS, and JavaScript with mouse event handlers that allows users to draw shapes (freehand, line, rectangle, ellipse), manipulate stroke properties, toggle grids/guides, and export drawings.

## Theory

* SVG (Scalable Vector Graphics):  
  A markup-based graphics format that defines vector-based shapes, paths, and lines. It is resolution-independent and easily styled or manipulated with CSS and JavaScript.
* Mouse and Pointer Events:  
  JavaScript provides events like pointerdown, pointermove, and pointerup that capture user interactions. These events are mapped to SVG coordinates to dynamically create and update shapes.
* Drawing Process:
  1. PointerDown: Detects the starting point of drawing and creates a new SVG shape.
  2. PointerMove: Updates the dimensions or path of the shape in real-time as the mouse/touch moves.
  3. PointerUp: Finalizes the shape and stops drawing.
* Event Handling:  
  By combining event listeners with SVG manipulation, we can support freehand drawing, straight lines, rectangles, ellipses, and canvas panning.
* Additional Features:
  1. Undo and Clear functionality via stack storage.
  2. Export options (SVG and rasterized PNG).
  3. Grid and guide overlays for precise drawing.
  4. Keyboard shortcuts for faster workflow.

**CODE:**

<!doctype html>

<html lang="en">

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width,initial-scale=1" />

<title>Interactive SVG Drawing Tool</title>

<style>

:root{--bg:#0f1724;--panel:#0b1220;--muted:#9aa4b2;--accent:#06b6d4}

html,body{height:100%;margin:0;font-family:Inter,system-ui,Segoe UI,Roboto,Arial}

body{display:flex;gap:12px;padding:12px;background:linear-gradient(180deg,#071024, #07162a);color:#e6eef6}

.panel{width:320px;background:linear-gradient(180deg, rgba(255,255,255,0.02), rgba(255,255,255,0.01));border-radius:12px;padding:14px;box-shadow:0 6px 24px rgba(2,6,23,0.6)}

.canvas-wrap{flex:1;border-radius:12px;overflow:hidden;display:flex;flex-direction:column}

.toolbar{display:flex;flex-wrap:wrap;gap:8px;margin-bottom:8px}

button,select,input[type=range]{background:transparent;border:1px solid rgba(255,255,255,0.06);color:inherit;padding:8px 10px;border-radius:8px;cursor:pointer}

.btn-active{border-color:var(--accent);box-shadow:0 6px 18px rgba(6,182,212,0.08)}

.svg-area{flex:1;background:linear-gradient(90deg, rgba(255,255,255,0.01) 0%, rgba(255,255,255,0.00) 100%);display:flex;align-items:stretch;justify-content:stretch}

svg{width:100%;height:100%;touch-action:none;user-select:none}

.status{font-size:13px;color:var(--muted);margin-top:10px}

.control-row{display:flex;gap:6px;align-items:center;margin-bottom:8px}

label{font-size:13px;color:var(--muted);min-width:110px}

input[type=color]{width:40px;height:40px;padding:0;border-radius:8px}

.small{padding:6px 8px;font-size:13px}

footer{font-size:12px;color:#86a0b8;margin-top:12px}

</style>

</head>

<body>

<aside class="panel">

<h3>SVG Drawing Tool — Bytexl Nimbus</h3>

<div class="control-row">

<label>Tool</label>

<div class="toolbar" role="toolbar" aria-label="tools">

<button id="tool-pen" class="btn small btn-active" title="Freehand (P)">Pen</button>

<button id="tool-line" class="btn small" title="Line (L)">Line</button>

<button id="tool-rect" class="btn small" title="Rectangle (R)">Rect</button>

<button id="tool-ellipse" class="btn small" title="Ellipse (E)">Ellipse</button>

<button id="tool-move" class="btn small" title="Pan/Move (Space)">Pan</button>

</div>

</div>

<div class="control-row">

<label>Stroke</label>

<input id="stroke-width" type="range" min="1" max="40" value="3">

<span id="stroke-value" style="min-width:26px;text-align:center">3</span>

</div>

<div class="control-row">

<label>Color</label>

<input id="color" type="color" value="#06b6d4">

<select id="linecap" title="Line cap">

<option value="round">Round</option>

<option value="square">Square</option>

<option value="butt">Butt</option>

</select>

</div>

<div class="control-row">

<label>Actions</label>

<div style="display:flex;gap:6px;flex-wrap:wrap">

<button id="undo">Undo</button>

<button id="clear">Clear</button>

<button id="download-svg">Download SVG</button>

<button id="download-png">Export PNG</button>

</div>

</div>

<div class="control-row">

<label>Options</label>

<div style="display:flex;gap:6px">

<button id="toggle-grid">Grid</button>

<button id="toggle-guides">Guides</button>

</div>

</div>

<div class="status" id="status">Ready — Pen selected</div>

<footer>Shortcuts: P Pen · L Line · R Rect · E Ellipse · Z Undo · C Clear</footer>

</aside>

<main class="canvas-wrap">

<div style="padding:8px;background:transparent">

<div style="display:flex;gap:8px;align-items:center">

<strong>Canvas</strong>

<small style="color:var(--muted)">(pointer/touch supported)</small>

</div>

</div>

<div class="svg-area">

<svg id="svgroot" xmlns="http://www.w3.org/2000/svg" viewBox="0 0 1200 800" preserveAspectRatio="xMidYMid meet">

<defs>

<pattern id="grid" width="40" height="40" patternUnits="userSpaceOnUse">

<path d="M 40 0 L 0 0 0 40" fill="none" stroke="rgba(255,255,255,0.03)" stroke-width="1"/>

</pattern>

</defs>

<rect id="bg" x="0" y="0" width="1200" height="800" fill="url(#grid)"></rect>

<g id="guides" opacity="0.6"></g>

<g id="drawing-layer"></g>

</svg>

</div>

</main>

<script>

// Helpers

const $ = id => document.getElementById(id);

const svg = $('svgroot');

const layer = $('drawing-layer');

const bg = $('bg');

const guides = $('guides');

// State

let tool = 'pen';

let drawing = false;

let current = null; // current SVG element being drawn

let startPoint = {x:0,y:0};

let stroke = $('#stroke-width');

let strokeValue = $('#stroke-value');

let colorPicker = $('#color');

let undoStack = [];

const MAX\_UNDO = 50;

let isPanning = false;

let viewBox = {x:0,y:0,w:1200,h:800};

// Coordinate conversion: pointer event to SVG coords

function getPointFromEvent(evt){

const pt = svg.createSVGPoint();

pt.x = evt.clientX; pt.y = evt.clientY;

const ctm = svg.getScreenCTM().inverse();

const svgP = pt.matrixTransform(ctm);

return {x: svgP.x, y: svgP.y};

}

// Tool buttons

function setActiveTool(t){

tool = t;

document.querySelectorAll('[id^="tool-"]').forEach(b=>b.classList.remove('btn-active'));

const btn = document.getElementById('tool-'+t);

if(btn) btn.classList.add('btn-active');

$('status').textContent = `Ready — ${t[0].toUpperCase()+t.slice(1)} selected`;

}

// Create an SVG element helper

function svgEl(tag, attrs={}){

const el = document.createElementNS('http://www.w3.org/2000/svg', tag);

for(const k in attrs) el.setAttribute(k, attrs[k]);

return el;

}

// Snapshot for undo

function pushUndo(){

undoStack.push(layer.innerHTML);

if(undoStack.length>MAX\_UNDO) undoStack.shift();

}

// Begin drawing

function pointerDown(e){

if(e.pointerType === 'mouse' && e.button !== 0) return; // left only

const p = getPointFromEvent(e);

drawing = true;

startPoint = p;

pushUndo();

if(tool === 'pen'){

const path = svgEl('path', {d:`M ${p.x} ${p.y}`, fill:'none', stroke:colorPicker.value, 'stroke-width':stroke.value, 'stroke-linecap':$('#linecap').value, 'stroke-linejoin':'round'});

layer.appendChild(path);

current = path;

} else if(tool === 'line'){

const line = svgEl('line', {x1:p.x, y1:p.y, x2:p.x, y2:p.y, stroke:colorPicker.value, 'stroke-width':stroke.value, 'stroke-linecap':$('#linecap').value});

layer.appendChild(line);

current = line;

} else if(tool === 'rect'){

const rect = svgEl('rect', {x:p.x, y:p.y, width:0, height:0, fill:'none', stroke:colorPicker.value, 'stroke-width':stroke.value});

layer.appendChild(rect);

current = rect;

} else if(tool === 'ellipse'){

const ell = svgEl('ellipse', {cx:p.x, cy:p.y, rx:0, ry:0, fill:'none', stroke:colorPicker.value, 'stroke-width':stroke.value});

layer.appendChild(ell);

current = ell;

} else if(tool === 'move'){

isPanning = true;

panStart = {x:e.clientX, y:e.clientY, viewBox:{...viewBox}};

}

svg.setPointerCapture(e.pointerId);

}

// Move

function pointerMove(e){

if(!drawing) return;

const p = getPointFromEvent(e);

if(!current) return;

if(tool === 'pen'){

const d = current.getAttribute('d');

current.setAttribute('d', d + ` L ${p.x} ${p.y}`);

} else if(tool === 'line'){

current.setAttribute('x2', p.x);

current.setAttribute('y2', p.y);

} else if(tool === 'rect'){

const x = Math.min(startPoint.x, p.x);

const y = Math.min(startPoint.y, p.y);

const w = Math.abs(p.x - startPoint.x);

const h = Math.abs(p.y - startPoint.y);

current.setAttribute('x', x);

current.setAttribute('y', y);

current.setAttribute('width', w);

current.setAttribute('height', h);

} else if(tool === 'ellipse'){

const rx = Math.abs(p.x - startPoint.x)/2;

const ry = Math.abs(p.y - startPoint.y)/2;

const cx = (p.x + startPoint.x)/2;

const cy = (p.y + startPoint.y)/2;

current.setAttribute('cx', cx);

current.setAttribute('cy', cy);

current.setAttribute('rx', rx);

current.setAttribute('ry', ry);

} else if(tool === 'move' && isPanning){

// simple pan implementation by adjusting viewBox

const dx = (panStart.x - e.clientX) \* (viewBox.w / svg.clientWidth);

const dy = (panStart.y - e.clientY) \* (viewBox.h / svg.clientHeight);

viewBox.x = panStart.viewBox.x + dx;

viewBox.y = panStart.viewBox.y + dy;

svg.setAttribute('viewBox', `${viewBox.x} ${viewBox.y} ${viewBox.w} ${viewBox.h}`);

}

}

// End drawing

function pointerUp(e){

if(!drawing) return;

drawing = false;

current = null;

isPanning = false;

}

// Wire events

svg.addEventListener('pointerdown', pointerDown);

svg.addEventListener('pointermove', pointerMove);

svg.addEventListener('pointerup', pointerUp);

svg.addEventListener('pointercancel', pointerUp);

// Tool buttons events

document.getElementById('tool-pen').addEventListener('click', ()=>setActiveTool('pen'));

document.getElementById('tool-line').addEventListener('click', ()=>setActiveTool('line'));

document.getElementById('tool-rect').addEventListener('click', ()=>setActiveTool('rect'));

document.getElementById('tool-ellipse').addEventListener('click', ()=>setActiveTool('ellipse'));

document.getElementById('tool-move').addEventListener('click', ()=>setActiveTool('move'));

// UI controls

stroke.addEventListener('input', ()=>{ strokeValue.textContent = stroke.value; });

$('#undo').addEventListener('click', ()=>{

if(undoStack.length>0){

const last = undoStack.pop();

layer.innerHTML = last || '';

}

});

$('#clear').addEventListener('click', ()=>{ pushUndo(); layer.innerHTML=''; });

// Grid toggle

let gridOn = true;

$('#toggle-grid').addEventListener('click', ()=>{ gridOn = !gridOn; bg.setAttribute('fill', gridOn? 'url(#grid)' : '#081321'); });

// Guides sample

let guidesOn = false;

$('#toggle-guides').addEventListener('click', ()=>{

guidesOn = !guidesOn;

guides.innerHTML = guidesOn ? `

<line x1="0" y1="400" x2="1200" y2="400" stroke="rgba(255,255,255,0.04)" stroke-dasharray="6 6" />

<line x1="600" y1="0" x2="600" y2="800" stroke="rgba(255,255,255,0.04)" stroke-dasharray="6 6" />

` : '';

});

// Download SVG

function download(filename, content){

const a = document.createElement('a');

a.href = content;

a.download = filename;

document.body.appendChild(a);

a.click();

a.remove();

}

$('#download-svg').addEventListener('click', ()=>{

const serializer = new XMLSerializer();

// Include drawing content into a fresh svg wrapper for portability

const clone = svg.cloneNode(true);

// remove pointer-capture data if any

clone.removeAttribute('style');

const xml = serializer.serializeToString(clone);

const svgBlob = new Blob([xml], {type: 'image/svg+xml;charset=utf-8'});

const url = URL.createObjectURL(svgBlob);

download('drawing.svg', url);

setTimeout(()=>URL.revokeObjectURL(url), 10000);

});

// Export PNG by rasterizing the SVG

$('#download-png').addEventListener('click', async ()=>{

const serializer = new XMLSerializer();

const clone = svg.cloneNode(true);

const xml = serializer.serializeToString(clone);

const svg64 = btoa(unescape(encodeURIComponent(xml)));

const image64 = 'data:image/svg+xml;base64,'+svg64;

const img = new Image();

img.onload = ()=>{

const canvas = document.createElement('canvas');

canvas.width = svg.viewBox.baseVal.width;

canvas.height = svg.viewBox.baseVal.height;

const ctx = canvas.getContext('2d');

// white background

ctx.fillStyle = '#ffffff';

ctx.fillRect(0,0,canvas.width,canvas.height);

ctx.drawImage(img,0,0);

const png = canvas.toDataURL('image/png');

download('drawing.png', png);

};

img.onerror = (err)=>{ alert('Failed to export PNG.'); };

img.src = image64;

});

// Keyboard shortcuts

window.addEventListener('keydown', (e)=>{

if(e.key === 'p' || e.key === 'P') setActiveTool('pen');

if(e.key === 'l' || e.key === 'L') setActiveTool('line');

if(e.key === 'r' || e.key === 'R') setActiveTool('rect');

if(e.key === 'e' || e.key === 'E') setActiveTool('ellipse');

if(e.key === ' ') { e.preventDefault(); setActiveTool('move'); }

if(e.key === 'z' || (e.ctrlKey && e.key === 'z')) document.getElementById('undo').click();

if(e.key === 'c' || e.key === 'C') document.getElementById('clear').click();

});

// keep stroke and color in sync for new shapes

const observer = new MutationObserver(()=>{

document.querySelectorAll('#drawing-layer [stroke]').forEach(el=>{

// nothing by default: shapes keep their original style

});

});

observer.observe(layer, {childList:true, subtree:true});

// Initialize

strokeValue.textContent = stroke.value;

svg.setAttribute('viewBox', `${viewBox.x} ${viewBox.y} ${viewBox.w} ${viewBox.h}`);

// Accessibility: make toolbar buttons focusable

document.querySelectorAll('button').forEach(b=>b.setAttribute('tabindex', '0'));

</script>

</body>

</html>

