**Name: Nikhil Sharma**

**UID : 23BCC70030**

**Class: 23BCC-1(A)**

**Experiment-2.3**

**Aim**

To design and implement an interactive SVG drawing tool using HTML, CSS, and JavaScript that allows users to draw, edit, and manipulate shapes using mouse event handlers.

**Theory**

SVG (Scalable Vector Graphics) is an XML-based format used to represent 2D graphics.  
Using SVG with JavaScript event listeners (mousedown, mousemove, mouseup), we can capture user interactions and dynamically update graphical elements.

**Key Concepts:**

* **SVG Elements**: rect, circle, ellipse, line, and path can be created and modified.
* **Mouse Events**:
  + mousedown → Start drawing/selection
  + mousemove → Update the shape as the cursor moves
  + mouseup → Finalize the shape
* **State Management**: Keeps track of the current tool, selected element, and drawing status.
* **Interactivity**: Features like stroke color, fill color, grid snapping, undo/redo, and export are added for a complete drawing experience.

**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 (Single File)</title>

<style>

:root{

--bg:#0b0f14; /\* deep slate \*/

--panel:#0f1520; /\* inky \*/

--muted:#94a3b8; /\* slate-400 \*/

--text:#e6edf3; /\* soft white \*/

--brand:#7c3aed; /\* violet \*/

--accent:#22c55e; /\* green \*/

--danger:#ef4444; /\* red \*/

--outline:#60a5fa; /\* sky \*/

}

\* { box-sizing: border-box; }

html, body {

height: 100%;

margin: 0;

font-family: ui-sans-serif, system-ui, -apple-system, Segoe UI, Roboto, Ubuntu, Cantarell, Noto Sans, Helvetica Neue, Arial, "Apple Color Emoji", "Segoe UI Emoji";

color: var(--text);

background: radial-gradient(1200px 800px at 80% -10%, #18202b 0%, var(--bg) 55%);

}

.app {

display: grid;

grid-template-rows: auto 1fr auto;

grid-template-columns: 280px 1fr;

grid-template-areas:

"toolbar toolbar"

"sidebar stage"

"status status";

height: 100svh;

gap: 10px;

padding: 10px;

}

.toolbar {

grid-area: toolbar;

display: flex;

flex-wrap: wrap;

align-items: center;

gap: 8px;

padding: 10px;

background: color-mix(in lab, var(--panel), black 12%);

border: 1px solid #1f2a37;

border-radius: 16px;

box-shadow: 0 6px 24px rgba(0,0,0,.35);

backdrop-filter: blur(8px);

}

.sidebar {

grid-area: sidebar;

display: grid;

grid-template-rows: auto 1fr;

gap: 10px;

min-height: 0;

}

.card {

background: color-mix(in lab, var(--panel), black 6%);

border: 1px solid #1f2a37;

border-radius: 16px;

padding: 12px;

box-shadow: 0 8px 24px rgba(0,0,0,.35);

}

.stage-wrap {

grid-area: stage;

position: relative;

min-height: 0;

}

.stage {

position: absolute;

inset: 0;

border-radius: 16px;

overflow: hidden;

border: 1px solid #1f2a37;

background: #0b1016;

touch-action: none; /\* avoid scrolling during draw on mobile \*/

}

.grid {

position: absolute; inset: 0; pointer-events: none;

background-image:

linear-gradient(to right, rgba(255,255,255,.06) 1px, transparent 1px),

linear-gradient(to bottom, rgba(255,255,255,.06) 1px, transparent 1px);

background-size: var(--gridsize, 20px) var(--gridsize, 20px);

mask: linear-gradient(180deg, #000 80%, transparent);

}

.btn {

display: inline-flex; align-items: center; gap: 8px;

height: 36px; padding: 0 12px; border-radius: 12px;

background: #121a25; color: var(--text); border: 1px solid #1e293b;

cursor: pointer; user-select: none;

transition: transform .06s ease, background .2s ease, border .2s ease;

}

.btn[aria-pressed="true"], .btn.primary { background: var(--brand); border-color: #5b21b6; }

.btn:hover { background: #162132; }

.btn:active { transform: translateY(1px); }

.btn.danger { background: #1a1111; border-color: #3b1d1d; }

.row { display: flex; align-items: center; gap: 8px; flex-wrap: wrap; }

.grow { flex: 1 1 auto; }

label { font-size: 12px; color: var(--muted); }

input[type="color"], select, input[type="number"], input[type="range"] {

height: 36px; border-radius: 10px; background: #0f1622; color: var(--text);

border: 1px solid #1f2a37; padding: 0 8px; outline: none; min-width: 0;

}

input[type="range"] { padding: 0; width: 140px; }

.status { grid-area: status; font-size: 12px; color: var(--muted); padding: 6px 2px; display: flex; gap: 14px; flex-wrap: wrap; }

/\* SVG styles \*/

svg { width: 100%; height: 100%; display: block; background: transparent; }

.handle-ring { fill: none; stroke: rgba(255,255,255,.18); stroke-width: 1; pointer-events: none; }

.selected { filter: drop-shadow(0 0 0.6px rgba(96,165,250,.9)); }

.selection-outline { pointer-events: none; fill: none; stroke: var(--outline); stroke-width: 1.2; stroke-dasharray: 4 3; }

.kbd { background:#0d1522; border:1px solid #1f2a37; padding:2px 6px; border-radius:6px; font-family: ui-monospace, SFMono-Regular, Menlo, Monaco, Consolas, "Liberation Mono", "Courier New", monospace; font-size: 12px; }

@media (max-width: 1100px) {

.app { grid-template-columns: 1fr; grid-template-areas:

"toolbar"

"stage"

"sidebar"

"status"; }

}

</style>

</head>

<body>

<div class="app">

<!-- Toolbar -->

<div class="toolbar" id="toolbar">

<button class="btn" data-tool="select" aria-pressed="true" title="Select (V)">🖱️ Select</button>

<button class="btn" data-tool="pen" title="Freehand Pen (P)">✏️ Pen</button>

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

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

<button class="btn" data-tool="ellipse" title="Ellipse (O)">◯ Ellipse</button>

<div style="width:1px;height:28px;background:#233046;margin:0 6px"></div>

<label>Stroke <input id="stroke" type="color" value="#e6edf3" /></label>

<label>Fill <input id="fill" type="color" value="#00000000" /></label>

<label>Width <input id="width" type="range" min="1" max="20" step="1" value="2" /></label>

<div class="grow"></div>

<button class="btn" id="undo" title="Undo (Ctrl/⌘+Z)">↶ Undo</button>

<button class="btn" id="redo" title="Redo (Ctrl/⌘+Y or Ctrl/⌘+Shift+Z)">↷ Redo</button>

<button class="btn" id="clear" title="Clear Canvas">🧹 Clear</button>

<button class="btn" id="download" title="Download SVG">⬇️ Export</button>

<label class="row" style="margin-left:8px"><input id="snap" type="checkbox" /> Snap</label>

<label class="row"><input id="showGrid" type="checkbox" checked /> Grid</label>

</div>

<!-- Sidebar -->

<div class="sidebar">

<div class="card">

<div class="row" style="justify-content: space-between; align-items: baseline;">

<strong>Inspector</strong>

<span id="inspectorType" class="kbd">—</span>

</div>

<div class="row" style="margin-top:8px; gap:12px">

<label> x <input type="number" id="propX" style="width:90px" /></label>

<label> y <input type="number" id="propY" style="width:90px" /></label>

<label> w <input type="number" id="propW" style="width:90px" /></label>

<label> h <input type="number" id="propH" style="width:90px" /></label>

</div>

<div class="row" style="margin-top:8px; gap:12px">

<button class="btn" id="bringFront" title="Bring to Front">⬆️ Front</button>

<button class="btn" id="sendBack" title="Send to Back">⬇️ Back</button>

<button class="btn danger" id="delete" title="Delete (Del)">🗑️ Delete</button>

</div>

<div style="margin-top:8px; font-size:12px; color:var(--muted)">

Shortcuts: <span class="kbd">V</span> select, <span class="kbd">P</span> pen, <span class="kbd">L</span> line, <span class="kbd">R</span> rect, <span class="kbd">O</span> ellipse, <span class="kbd">Ctrl/⌘+Z</span> undo, <span class="kbd">Del</span> delete, hold <span class="kbd">Shift</span> to constrain (line 45°, circle, square).

</div>

</div>

<div class="card" style="overflow:auto;">

<strong>Layers</strong>

<ol id="layers" style="margin:8px 0 0 18px; padding:0; line-height:1.7"></ol>

</div>

</div>

<!-- Stage -->

<div class="stage-wrap">

<div class="stage" id="stage">

<div class="grid" id="grid"></div>

<svg id="svg" xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" viewBox="0 0 2000 1200">

<g id="content"></g>

<rect id="selectionRect" class="selection-outline" visibility="hidden" />

</svg>

</div>

</div>

<!-- Status Bar -->

<div class="status" id="status">Ready.</div>

</div>

<script>

// ===== Utilities =====

const $ = (sel, root=document) => root.querySelector(sel);

const $$ = (sel, root=document) => Array.from(root.querySelectorAll(sel));

const svgNS = 'http://www.w3.org/2000/svg';

const ix = v => parseFloat(v || 0) || 0;

function clamp(n, min, max){ return Math.max(min, Math.min(max, n)); }

function snap(n, g){ return Math.round(n / g) \* g; }

function pt(svg, clientX, clientY){

const p = svg.createSVGPoint(); p.x = clientX; p.y = clientY;

return p.matrixTransform(svg.getScreenCTM().inverse());

}

// ===== App State =====

const state = {

tool: 'select',

drawing: null, // current element being drawn

start: null, // {x,y}

current: null, // {x,y}

selected: null,

drag: null, // {dx, dy}

history: [],

future: [],

grid: 20,

isDown: false,

};

const els = {

svg: $('#svg'),

g: $('#content'),

selectionRect: $('#selectionRect'),

stage: $('#stage'),

grid: $('#grid'),

toolbar: $('#toolbar'),

status: $('#status'),

layers: $('#layers'),

inspectorType: $('#inspectorType'),

propX: $('#propX'), propY: $('#propY'), propW: $('#propW'), propH: $('#propH'),

stroke: $('#stroke'), fill: $('#fill'), width: $('#width'),

undo: $('#undo'), redo: $('#redo'), clear: $('#clear'), download: $('#download'),

delete: $('#delete'), bringFront: $('#bringFront'), sendBack: $('#sendBack'),

snap: $('#snap'), showGrid: $('#showGrid'),

};

// Initialize

els.grid.style.setProperty('--gridsize', state.grid + 'px');

// ===== History =====

function pushHistory(){

state.history.push(els.g.innerHTML);

if (state.history.length > 300) state.history.shift();

state.future.length = 0;

updateButtons();

}

function undo(){ if(!state.history.length) return; state.future.push(els.g.innerHTML); els.g.innerHTML = state.history.pop(); clearSelection(); rebuildLayers(); announce('Undone.'); }

function redo(){ if(!state.future.length) return; state.history.push(els.g.innerHTML); els.g.innerHTML = state.future.pop(); clearSelection(); rebuildLayers(); announce('Redone.'); }

// ===== UI helpers =====

function setTool(tool){

state.tool = tool; $$('.toolbar .btn').forEach(b=>b.setAttribute('aria-pressed', String(b.dataset.tool===tool)));

announce(`Tool: ${tool}`);

}

function announce(msg){ els.status.textContent = msg; }

function updateButtons(){

els.undo.disabled = state.history.length === 0;

els.redo.disabled = state.future.length === 0;

}

function clearSelection(){

if (state.selected){ state.selected.classList.remove('selected'); }

state.selected = null; updateInspector();

els.selectionRect.setAttribute('visibility', 'hidden');

}

function select(el){

if(!el) return clearSelection();

if (state.selected) state.selected.classList.remove('selected');

state.selected = el; el.classList.add('selected');

updateInspector();

drawSelectionOutline();

}

function drawSelectionOutline(){

const el = state.selected; if(!el) return;

const bbox = el.getBBox();

els.selectionRect.setAttribute('x', bbox.x);

els.selectionRect.setAttribute('y', bbox.y);

els.selectionRect.setAttribute('width', bbox.width);

els.selectionRect.setAttribute('height', bbox.height);

els.selectionRect.setAttribute('visibility', 'visible');

}

function updateInspector(){

const el = state.selected; if(!el){

els.inspectorType.textContent = '—';

els.propX.value = ''; els.propY.value = ''; els.propW.value = ''; els.propH.value = '';

return;

}

els.inspectorType.textContent = el.tagName;

const bbox = el.getBBox();

els.propX.value = Math.round(bbox.x);

els.propY.value = Math.round(bbox.y);

els.propW.value = Math.round(bbox.width);

els.propH.value = Math.round(bbox.height);

}

function rebuildLayers(){

els.layers.innerHTML = '';

Array.from(els.g.children).forEach((child, i) => {

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

const name = child.getAttribute('data-name') || `${child.tagName} ${i+1}`;

li.textContent = name;

li.style.cursor = 'pointer';

li.onclick = () => select(child);

els.layers.appendChild(li);

});

}

// ===== Creation helpers =====

function applyStyle(el){

el.setAttribute('stroke', els.stroke.value);

el.setAttribute('fill', els.fill.value);

el.setAttribute('stroke-width', els.width.value);

el.setAttribute('vector-effect', 'non-scaling-stroke');

el.setAttribute('data-name', el.tagName);

}

function startDraw(x, y){

const snapOn = els.snap.checked;

const gs = state.grid;

const sx = snapOn ? snap(x, gs) : x;

const sy = snapOn ? snap(y, gs) : y;

state.start = {x: sx, y: sy};

state.current = {x: sx, y: sy};

let el;

if (state.tool === 'pen'){

el = document.createElementNS(svgNS, 'path');

el.\_d = [`M ${sx} ${sy}`];

el.setAttribute('d', el.\_d.join(' '));

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

el = document.createElementNS(svgNS, 'line');

el.setAttribute('x1', sx); el.setAttribute('y1', sy);

el.setAttribute('x2', sx); el.setAttribute('y2', sy);

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

el = document.createElementNS(svgNS, 'rect');

el.setAttribute('x', sx); el.setAttribute('y', sy);

el.setAttribute('width', 0); el.setAttribute('height', 0);

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

el = document.createElementNS(svgNS, 'ellipse');

el.setAttribute('cx', sx); el.setAttribute('cy', sy);

el.setAttribute('rx', 0); el.setAttribute('ry', 0);

}

if (el){ applyStyle(el); els.g.appendChild(el); state.drawing = el; }

}

function updateDraw(x, y, shiftKey){

if (!state.drawing) return;

const snapOn = els.snap.checked; const gs = state.grid;

let nx = snapOn ? snap(x, gs) : x;

let ny = snapOn ? snap(y, gs) : y;

if (state.tool === 'pen'){

const last = state.drawing.\_d[state.drawing.\_d.length-1];

const [\_, lx, ly] = /M|L\s+([\d.]+)\s+([\d.]+)/.exec(last) || [0, nx, ny];

const dx = nx - +lx, dy = ny - +ly;

const dist2 = dx\*dx + dy\*dy;

if (dist2 > 4) { state.drawing.\_d.push(`L ${nx} ${ny}`); state.drawing.setAttribute('d', state.drawing.\_d.join(' ')); }

}

else if (state.tool === 'line'){

if (shiftKey){ // constrain 0/45/90

const dx = nx - state.start.x; const dy = ny - state.start.y;

const ang = Math.atan2(dy, dx);

const step = Math.PI/4; // 45°

const snapped = Math.round(ang/step)\*step;

const len = Math.hypot(dx, dy);

nx = state.start.x + Math.cos(snapped)\*len;

ny = state.start.y + Math.sin(snapped)\*len;

}

state.drawing.setAttribute('x2', nx);

state.drawing.setAttribute('y2', ny);

}

else if (state.tool === 'rect'){

let rx = nx - state.start.x; let ry = ny - state.start.y;

if (shiftKey) { const s = Math.min(Math.abs(rx), Math.abs(ry)); rx = Math.sign(rx)\*s; ry = Math.sign(ry)\*s; }

const x0 = rx<0 ? nx : state.start.x;

const y0 = ry<0 ? ny : state.start.y;

state.drawing.setAttribute('x', x0);

state.drawing.setAttribute('y', y0);

state.drawing.setAttribute('width', Math.abs(rx));

state.drawing.setAttribute('height', Math.abs(ry));

}

else if (state.tool === 'ellipse'){

let rx = Math.abs(nx - state.start.x);

let ry = Math.abs(ny - state.start.y);

if (shiftKey) { const s = Math.min(rx, ry); rx = ry = s; }

const cx = (nx + state.start.x)/2; const cy = (ny + state.start.y)/2;

state.drawing.setAttribute('cx', cx);

state.drawing.setAttribute('cy', cy);

state.drawing.setAttribute('rx', rx/2);

state.drawing.setAttribute('ry', ry/2);

}

}

function endDraw(){

if (!state.drawing) return;

if (state.tool !== 'select') pushHistory();

state.drawing = null; state.start = null; state.current = null;

rebuildLayers();

}

// ===== Dragging selected =====

function beginDrag(el, startX, startY){

const bbox = el.getBBox();

state.drag = { ox: startX - bbox.x, oy: startY - bbox.y };

}

function updateDrag(el, x, y){

const bbox = el.getBBox();

const nx = x - state.drag.ox; const ny = y - state.drag.oy;

const snapOn = els.snap.checked; const gs = state.grid;

const dx = (snapOn ? snap(nx, gs) : nx) - bbox.x;

const dy = (snapOn ? snap(ny, gs) : ny) - bbox.y;

moveElement(el, dx, dy);

drawSelectionOutline();

updateInspector();

}

function endDrag(){ pushHistory(); state.drag = null; }

function moveElement(el, dx, dy){

const name = el.tagName;

if (name === 'rect'){

el.setAttribute('x', ix(el.getAttribute('x')) + dx);

el.setAttribute('y', ix(el.getAttribute('y')) + dy);

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

el.setAttribute('cx', ix(el.getAttribute('cx')) + dx);

el.setAttribute('cy', ix(el.getAttribute('cy')) + dy);

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

el.setAttribute('x1', ix(el.getAttribute('x1')) + dx);

el.setAttribute('y1', ix(el.getAttribute('y1')) + dy);

el.setAttribute('x2', ix(el.getAttribute('x2')) + dx);

el.setAttribute('y2', ix(el.getAttribute('y2')) + dy);

} else if (name === 'path'){

const parts = el.getAttribute('d').trim().split(/\s+/);

for (let i=0; i<parts.length; i++){

if (parts[i]==='M' || parts[i]==='L'){ const x = parseFloat(parts[i+1])+dx; const y = parseFloat(parts[i+2])+dy; parts[i+1]=x; parts[i+2]=y; i+=2; }

}

el.setAttribute('d', parts.join(' '));

}

}

// ===== Event Wiring =====

els.toolbar.addEventListener('click', (e)=>{

const b = e.target.closest('button[data-tool]'); if (!b) return;

setTool(b.dataset.tool);

});

els.svg.addEventListener('mousedown', (e)=>{

const p = pt(els.svg, e.clientX, e.clientY);

state.isDown = true;

if (state.tool === 'select'){

const target = e.target.closest('#content > \*');

if (target){ select(target); beginDrag(target, p.x, p.y); }

else { clearSelection(); }

} else {

startDraw(p.x, p.y);

}

e.preventDefault();

});

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

if (!state.isDown) return;

const p = pt(els.svg, e.clientX, e.clientY);

if (state.tool === 'select' && state.selected && state.drag){ updateDrag(state.selected, p.x, p.y); }

else { updateDraw(p.x, p.y, e.shiftKey); }

}, {passive:true});

window.addEventListener('mouseup', ()=>{

if (!state.isDown) return; state.isDown = false;

if (state.tool === 'select' && state.drag){ endDrag(); }

else { endDraw(); }

});

// ===== Inspector inputs =====

[els.propX, els.propY, els.propW, els.propH].forEach(inp=>{

inp.addEventListener('change', ()=>{

const el = state.selected; if (!el) return;

const tag = el.tagName; const x = +els.propX.value; const y = +els.propY.value; const w = +els.propW.value; const h = +els.propH.value;

if (tag==='rect'){ el.setAttribute('x', x); el.setAttribute('y', y); el.setAttribute('width', Math.max(0,w)); el.setAttribute('height', Math.max(0,h)); }

if (tag==='ellipse'){ el.setAttribute('cx', x + w/2); el.setAttribute('cy', y + h/2); el.setAttribute('rx', Math.max(0,w/2)); el.setAttribute('ry', Math.max(0,h/2)); }

if (tag==='line'){ el.setAttribute('x1', x); el.setAttribute('y1', y); el.setAttribute('x2', x+w); el.setAttribute('y2', y+h); }

if (tag==='path'){ /\* set via bbox not trivial; skip precise change \*/ }

drawSelectionOutline(); pushHistory();

});

});

// Style controls

[els.stroke, els.fill, els.width].forEach(inp=>{

inp.addEventListener('change', ()=>{

if (state.selected){

if (inp===els.stroke) state.selected.setAttribute('stroke', inp.value);

if (inp===els.fill) state.selected.setAttribute('fill', inp.value);

if (inp===els.width) state.selected.setAttribute('stroke-width', inp.value);

pushHistory(); drawSelectionOutline(); rebuildLayers();

}

});

});

// Layer ordering

els.bringFront.onclick = ()=>{ const el=state.selected; if(!el) return; els.g.appendChild(el); rebuildLayers(); pushHistory(); };

els.sendBack.onclick = ()=>{ const el=state.selected; if(!el) return; els.g.insertBefore(el, els.g.firstChild); rebuildLayers(); pushHistory(); };

// Delete

function deleteSelected(){ if (!state.selected) return; state.selected.remove(); clearSelection(); rebuildLayers(); pushHistory(); }

els.delete.onclick = deleteSelected;

// Undo/Redo/Clear/Export

els.undo.onclick = undo; els.redo.onclick = redo;

els.clear.onclick = ()=>{ if (!els.g.children.length) return; pushHistory(); els.g.innerHTML=''; clearSelection(); rebuildLayers(); announce('Cleared.'); };

els.download.onclick = ()=>{

const copy = els.svg.cloneNode(true);

copy.querySelector('#selectionRect')?.remove();

const xml = new XMLSerializer().serializeToString(copy);

const blob = new Blob([xml], {type:'image/svg+xml'});

const a = document.createElement('a'); a.href = URL.createObjectURL(blob); a.download = 'drawing.svg'; a.click();

setTimeout(()=>URL.revokeObjectURL(a.href), 5000);

};

// Grid & Snap

els.showGrid.onchange = ()=>{ els.grid.style.display = els.showGrid.checked ? 'block':'none'; };

// Keyboard shortcuts

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

const mod = e.ctrlKey || e.metaKey;

if (mod && e.key.toLowerCase()==='z'){ e.preventDefault(); if (e.shiftKey) redo(); else undo(); }

else if (mod && e.key.toLowerCase()==='y'){ e.preventDefault(); redo(); }

else if (e.key==='Delete'){ e.preventDefault(); deleteSelected(); }

else if (!mod){

const k = e.key.toLowerCase();

if (k==='v') setTool('select');

if (k==='p') setTool('pen');

if (k==='l') setTool('line');

if (k==='r') setTool('rect');

if (k==='o') setTool('ellipse');

}

});

// Click to select from layer panel sync highlight

els.g.addEventListener('click', (e)=>{

if (state.tool!=='select') return; const target = e.target.closest('#content > \*'); if (target) select(target);

});

// Initial state

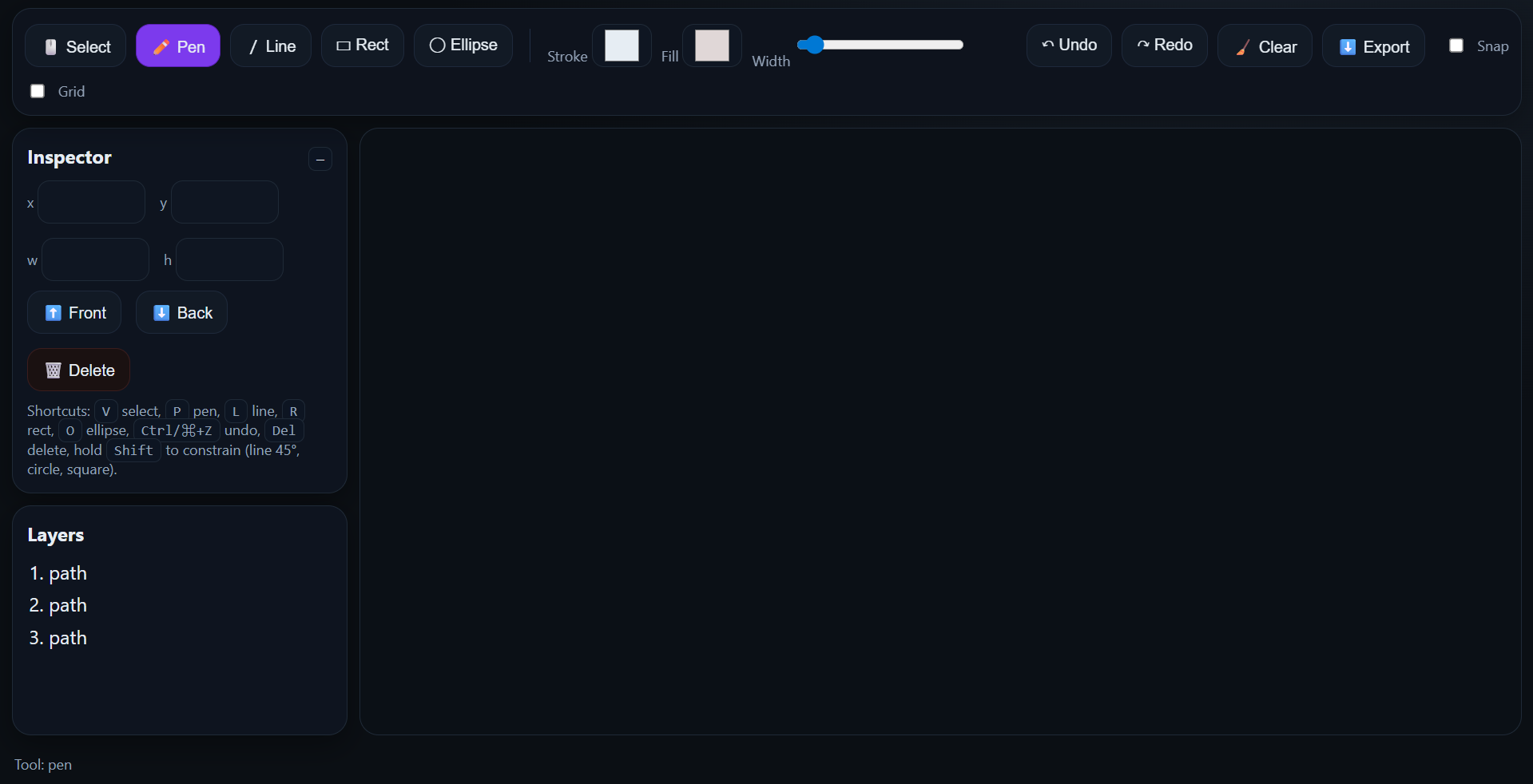
pushHistory(); rebuildLayers(); updateButtons();

</script>

</body>

</html>

**Output**



**Learning Outcomes**

After completing this experiment/project, the learner will be able to:

1. Understand how to manipulate **SVG elements** dynamically using JavaScript.
2. Apply **mouse event handling** to enable interactivity in graphics applications.
3. Implement **state management** for interactive tools.
4. Enhance user experience with features like **undo/redo, snapping, and exporting**.
5. Gain confidence in building **web-based design tools** using only HTML, CSS, and JavaScript.