```
document.addEventListener("DOMContentLoaded", () => {
        const canvas = document.getElementById('mazeCanvas');
        const ctx = canvas.getContext('2d');

        ctx.fillStyle = "#fdfd96";
        ctx.fillRect(23, 24, 750, 650);

        let canvasObjects = [
        { x: 100, y: 120, width: 80, height: 50 },
        { x: 300, y: 250, width: 100, height: 60 },
        { x: 500, y: 150, width: 70, height: 70 }
        ];

        function drawCanvasObjects() {
        canvasObjects.forEach(obj => {
        ctx.fillStyle = "#fdfd96";
        ctx.fillRect(obj.x, obj.y, obj.width, obj.height);
        });
        }

        console.log("startPoint reset");
        let startPoint = null;
        let drawnLines = [];
        let horizontalCutMode = false;
```

```
let verticalCutMode = false;

let slantCutMode = false;

let drawMode=true;

let blockNextClick = false;

let eraserMode = false;

let curvedCutMode = false;

let undoMode = false;


const horizontalCuts = [];

const verticalCuts = [];

const slantCuts = [];

let erasedPaths = [];

const curvedCuts = [];

let currentCurvePath = [];

let currentEraseStroke = [];

const actionHistory = [];

const placedObstacles = [];


document.getElementById("horizontalMaze").addEventListener("click", () => {

horizontalCutMode = true;

drawMode=false;

console.log("Horizontal cut mode activated");

});


document.getElementById("verticalMaze").addEventListener("click", () => {
```

```
verticalCutMode = true;
drawMode=false;
console.log("Vertical cut mode activated");
});

document.getElementById("slantMaze").addEventListener("click", () => {
slantCutMode = true;
drawMode = false;
console.log("Slant cut mode activated");
});

document.getElementById("curvedMaze").addEventListener("click", () => {
curvedCutMode = !curvedCutMode;

 if (curvedCutMode) {
        drawMode = false;
        canvas.style.cursor = 'crosshair';
        console.log("Curved cut mode activated");
 } else {
        drawMode = true;
        canvas.style.cursor = 'default';
        console.log("Curved cut mode deactivated");
 }

});
```

```
document.getElementById("eraserMaze").addEventListener("click", () => {
eraserMode = !eraserMode;

if (eraserMode) {
canvas.style.cursor = 'crosshair';
drawMode = false;
startPoint = null;
console.log("Eraser mode activated!");
} else {
canvas.style.cursor = 'default';
drawMode = true;
console.log("Eraser mode deactivated.");
}
});

document.getElementById("undoMaze").addEventListener("click", () => {
undoMode = true;
performUndo();
undoMode = false;
});

canvas.addEventListener("click", (event) => {
if (curvedCutMode) return;
```

```
if (undoMode) return;
const rect = canvas.getBoundingClientRect();
const x = event.clientX - rect.left;
const y = event.clientY - rect.top;
if (blockNextClick) {
blockNextClick = false;
return;
}



if (horizontalCutMode || verticalCutMode || slantCutMode) {
if (!startPoint) {
  startPoint = { x, y };
  ctx.fillStyle = "#ff1493";
  ctx.beginPath();
  ctx.arc(x, y, 3, 0, Math.PI * 2);
  ctx.fill();
  return;
} else {
  if (horizontalCutMode) {
            performHorizontalCut(startPoint.x, x, startPoint.y);
            horizontalCutMode = false;
  }
  if (verticalCutMode) {
            performVerticalCut(startPoint.y, y, startPoint.x);
```

```
            verticalCutMode = false;
  }
  if (slantCutMode) {
            performSlantCut(startPoint.x, startPoint.y, x, y);
            slantCutMode = false;
  }
  drawMode = true;
  redrawCuts();
  startPoint = null;
  return;
}
}

if (drawMode){
  if (!startPoint) {
            startPoint = { x, y };
            ctx.fillStyle = "#ff1439";
            ctx.beginPath();
            ctx.arc(x, y, 3, 0, Math.PI * 2);
            ctx.fill();
            return;
  }
  else {
            ctx.strokeStyle = "#ff0000";
            ctx.lineWidth = 3;
```

```
            ctx.beginPath();
            ctx.moveTo(startPoint.x, startPoint.y);
            ctx.lineTo(x, y);
            ctx.stroke();

            drawnLines.push({
              startX: startPoint.x,
              startY: startPoint.y,
              endX: x,
              endY: y
            });
            actionHistory.push({ type: "draw", data: drawnLines[drawnLines.length - 1] });
            startPoint = null;
  }
}
});

function performHorizontalCut(startX, endX, y) {
const cut = { startX, endX, y };
horizontalCuts.push(cut);
actionHistory.push({ type: "horizontal", data: cut });
}

function performVerticalCut(startY, endY, x) {
const cut = { startY, endY, x };
```

```
verticalCuts.push(cut);
actionHistory.push({ type: "vertical", data: cut });
}

function performSlantCut(startX, startY, endX, endY) {
const cut = { startX, startY, endX, endY };
slantCuts.push(cut);
actionHistory.push({ type: "slant", data: cut });
}

function redrawCuts() {
ctx.fillStyle = "#fdfd96";
ctx.fillRect(23, 24, 750, 650);

ctx.strokeStyle = "#ffffff";
ctx.lineWidth = 4;

drawCanvasObjects();

horizontalCuts.forEach(({ startX, endX, y }) => {
ctx.beginPath();
ctx.moveTo(startX, y);
ctx.lineTo(endX, y);
ctx.stroke();
});
```

```
verticalCuts.forEach(({ startY, endY, x }) => {
ctx.beginPath();
ctx.moveTo(x, startY);
ctx.lineTo(x, endY);
ctx.stroke();
});

slantCuts.forEach(({ startX, startY, endX, endY }) => {
ctx.beginPath();
ctx.moveTo(startX, startY);
ctx.lineTo(endX, endY);
ctx.stroke();
});

curvedCuts.forEach(path => {
ctx.strokeStyle = "#ffffff";
ctx.lineWidth = 3;
for (let i = 1; i < path.length; i++) {
ctx.beginPath();
ctx.moveTo(path[i - 1].x, path[i - 1].y);
ctx.lineTo(path[i].x, path[i].y);
ctx.stroke();
}
});
```

```
        drawnLines.forEach(line => {
        ctx.strokeStyle = "#ff0000";
        ctx.lineWidth = 3;
        ctx.beginPath();
        ctx.moveTo(line.startX, line.startY);
        ctx.lineTo(line.endX, line.endY);
        ctx.stroke();
        });

        erasedPaths.forEach(path => {
        path.forEach(({ x, y, radius }) => {
        ctx.fillStyle = "#ffffff";
        ctx.beginPath();
        ctx.arc(x, y, radius, 0, Math.PI * 2);
        ctx.fill();
        });
        });
        placedObstacles.forEach(({ name, x, y, width, height }) => {
  const img = obstacleImages[name];
  ctx.drawImage(img, x - width / 2, y - height / 2, width, height);
});

        }
```

```
canvas.addEventListener("mousedown", (e) => {
const { x, y } = getCursorPosition(e);

if (eraserMode) {
canvas.addEventListener("mousemove", handleEraser);
return;
}

if (curvedCutMode) {
currentCurvePath = [];
canvas.addEventListener("mousemove", traceCurve);
return;
}
});


canvas.addEventListener("mouseup", () => {
if (curvedCutMode) {
        canvas.removeEventListener("mousemove", traceCurve);
        curvedCuts.push([...currentCurvePath]);
        actionHistory.push({ type: "curve", data: [...currentCurvePath] });
        // store the full path
        currentCurvePath = [];
        if (currentCurvePath.length > 0) {
```

```
            curvedCuts.push([...currentCurvePath]);
            actionHistory.push({ type: "curve", data: [...currentCurvePath] });
            }
            currentCurvePath = [];
            blockNextClick = true;
            redrawCuts();
}

if (eraserMode){
  canvas.removeEventListener("mousemove", handleEraser);
  if (currentEraseStroke.length > 0) {
  erasedPaths.push([...currentEraseStroke]);
  actionHistory.push({ type: "erase", data: [...currentEraseStroke] });
  currentEraseStroke = [];
}
}
});

function traceCurve(e) {
const { x, y } = getCursorPosition(e);
currentCurvePath.push({ x, y });

ctx.strokeStyle = "#00ced1";
ctx.lineWidth = 3;
if (currentCurvePath.length > 1) {
```

```
const prev = currentCurvePath[currentCurvePath.length - 2];
ctx.beginPath();
ctx.moveTo(prev.x, prev.y);
ctx.lineTo(x, y);
ctx.stroke();
}
}

function getCursorPosition(e) {
const rect = canvas.getBoundingClientRect();
return {
x: e.clientX - rect.left,
y: e.clientY - rect.top
};
}

function performErase(path) {
const eraseRadius = 8; // adjust for how thick the erase is
path.forEach(point => {
ctx.fillStyle = "#ffffff"; // canvas base color
ctx.beginPath();
ctx.arc(point.x, point.y, eraseRadius, 0, Math.PI * 2);
ctx.fill();
});
}
```

```
function handleEraser(e) {
const { x, y } = getCursorPosition(e);
const eraseRadius = 12;

ctx.fillStyle = "#ffffff"; // erasing by painting white
ctx.beginPath();
ctx.arc(x, y, eraseRadius, 0, Math.PI * 2);
ctx.fill();

currentEraseStroke.push({ x, y, radius: eraseRadius });
}



function performUndo() {
if (actionHistory.length === 0) {
console.log("Nothing to undo");
return;
}

const lastAction = actionHistory.pop();

switch (lastAction.type) {
case "horizontal":
  horizontalCuts.pop();
```

```
        console.log("Undid horizontal cut");
        break;
      case "vertical":
        verticalCuts.pop();
        console.log("Undid vertical cut");
        break;
      case "slant":
        slantCuts.pop();
        console.log("Undid slant cut");
        break;
      case "curve":
        curvedCuts.pop();
        console.log("Undid curved cut");
        break;
      case "draw":
        drawnLines.pop();
        console.log("Undid drawn line");
        break;
      case "erase":
        erasedPaths.pop();
        console.log("Undid erase");
        break;
      case "obstacle":
placedObstacles.pop(); // ✅ remove last obstacle
break;
```

```
}

startPoint = null;
drawMode = true;
blockNextClick = false;
curvedCutMode = false;
canvas.style.cursor = 'default';
redrawCuts();
}
let draggedImageName = null;
const obstacleImages = {};

// Load obstacle images
document.querySelectorAll('.obstacle').forEach(img => {
  const name = img.dataset.name;
  const image = new Image();
  image.src = img.src;
  obstacleImages[name] = image;

  img.addEventListener('dragstart', () => {
            draggedImageName = name;
  });
});

// Allow drop on canvas
```

```
    canvas.addEventListener('dragover', e => e.preventDefault());

    canvas.addEventListener('drop', e => {
      e.preventDefault();
      if (!draggedImageName) return;

      const rect = canvas.getBoundingClientRect();
      const x = e.clientX - rect.left;
      const y = e.clientY - rect.top;

      const img = obstacleImages[draggedImageName];
      const fixedWidth = 70;
      const fixedHeight = 70;

      ctx.drawImage(img, x - fixedWidth / 2, y - fixedHeight / 2, fixedWidth, fixedHeight);


      // Optional: store for undo or game logic
      placedObstacles.push({
name: draggedImageName,
        x,
        y,
        width: 70,
        height: 70
      });
```

```
            actionHistory.push({ type: "obstacle", data: { name: draggedImageName, x, y } });


            draggedImageName = null;
          });
});
```