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

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

        let draggedImageName = null;
        const obstacleImages = {};
        const placedObstacles = [];
        const actionHistory = [];
        let selectedObstacle = null;
        let isDragging = false;
        let rotateMode = false;

        document.getElementById("undo").addEventListener("click", () => {
          performUndo();
        });

        // 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;
  });
});

//rotate button
document.getElementById("rotate").addEventListener("click", () => {
  rotateMode = !rotateMode;

  const btn = document.getElementById("rotate");
  btn.textContent = rotateMode ? "🛑 Stop Rotating" : "🔄 Rotate";
});

function drawRotatedImage(img, x, y, width, height, rotation) {
  ctx.save();
  ctx.translate(x, y);
  ctx.rotate((rotation * Math.PI) / 180);
  ctx.drawImage(img, -width / 2, -height / 2, width, height);
  ctx.restore();
}
// Allow drop on canvas
canvas.addEventListener('dragover', e => e.preventDefault());

canvas.addEventListener("mousedown", e => {
```

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

  // Check if click is inside any obstacle (reverse order for topmost)
  for (let i = placedObstacles.length - 1; i >= 0; i--) {
        const obj = placedObstacles[i];
        const dx = x - obj.x;
        const dy = y - obj.y;
        const w = obj.width;
        const h = obj.height;

        // Rough bounding box check (rotation not accounted yet)
        if (Math.abs(dx) < w / 2 && Math.abs(dy) < h / 2) {
          selectedObstacle = obj;
          isDragging = true;
          break;
        }
  }
});

canvas.addEventListener("mousemove", e => {
  if (!isDragging || !selectedObstacle) return;

  const rect = canvas.getBoundingClientRect();
```

```
  selectedObstacle.x = e.clientX - rect.left;
  selectedObstacle.y = e.clientY - rect.top;

  redrawCanvas();
});

canvas.addEventListener("mouseup", () => {
  isDragging = false;
  selectedObstacle = null;
});

canvas.addEventListener("click", e => {
  const rect = canvas.getBoundingClientRect();
  const x = e.clientX - rect.left;
  const y = e.clientY - rect.top;

  for (let i = placedObstacles.length - 1; i >= 0; i--) {
        const obj = placedObstacles[i];
        const dx = x - obj.x;
        const dy = y - obj.y;
        const w = obj.width;
        const h = obj.height;

        if (Math.abs(dx) < w / 2 && Math.abs(dy) < h / 2) {
          if (rotateMode) {
```

```
                obj.rotation = (obj.rotation + 45) % 360;
            } else {
                selectedObstacle = obj;
            }
            redrawCanvas();
            break;
          }
  }
});
function redrawCanvas() {
  ctx.clearRect(23, 24, 750, 650);
  ctx.fillStyle = "#fdfd96";
  ctx.fillRect(23, 24, 750, 650);

  placedObstacles.forEach(obj => {
        const img = obstacleImages[obj.name];
        drawRotatedImage(img, obj.x, obj.y, obj.width, obj.height, obj.rotation);
  });
}

//dropping images
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 width = img.naturalWidth;
  const height = img.naturalHeight;

  drawRotatedImage(img, x, y, width, height, 0);

  placedObstacles.push({
        name: draggedImageName,
        x,
        y,
        width,
        height,
        rotation: 0
  });

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

  draggedImageName = null;
});

function performUndo() {
```

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

        actionHistory.pop();
        placedObstacles.pop();

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

        placedObstacles.forEach(obj => {
            const img = obstacleImages[obj.name];
            drawRotatedImage(img, obj.x, obj.y, obj.width, obj.height, obj.rotation);
        });
    }
});
```