r/ThingsYouDidntKnow • 13 days ago
TheStocksGuy

Chuck-a-Puck: Puck placements HTML Canvas Playground

Welcome to the **Chuck-a-Dot** aka **Chuck-a-Puck** tutorial, inspired by the **The Lonely Runner Conjecture**. Here, explains how an animated donut-shaped canvas with dots (pucks) is created and how they move without overlapping.

Setting Up the Donut

Imagine a donut shape with a width of 150 units. Each puck (dot) is 7.5% of this width.

dot size is 0 point 075 times 150

Spiral Movement

Each puck starts at a unique angle and distance from the center, forming a spiral pattern.

angle equals index divided by total dots times two pi distance equals width divided by two

Where: index ranges from zero to forty-nine total dots equals fifty

Calculating Positions

To place each puck on the canvas, we calculate its x and y coordinates:

x equals center x plus distance times cosine of angle y equals center y plus distance times sine of angle

Avoiding Overlaps

Our pucks never overlap. We ensure they maintain a one-pixel gap.

For each pair of pucks i and j:

delta x equals x of puck i minus x of puck j delta y equals y of puck i minus y of puck j distance equals square root of delta x squared plus delta y squared

If distance is less than the dot size: Adjust positions based on puck locations relative to the canvas center: x of puck i plus or minus adjustment y of puck i plus or minus adjustment

Controlling Speed and Reset

You can control the speed of the pucks with an adjustable input called speed. Resetting will bring them back to their default pace.

Tracking Positions Every 60ms

Every sixty milliseconds, we log the positions of the pucks.

positions at time t equals x and y coordinates of each puck

Winning the Game

With our Chuck-a-Dot system, the pucks never overlap and stay within the donut boundaries, moving smoothly and swiftly.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Chuck-a-Dot</title>
  <style>
    body {
      display: flex;
      flex-direction: column;
      align-items: center;
      justify-content: center;
      height: 100vh;
      background-color: #000; /* Dark background */
      margin: 0;
    }
    canvas {
      border: 1px solid black;
      background-color: #fff; /* Light background for the canvas */
    }
    input {
      margin: 10px;
    }
  </style>
</head>
<body>
  <canvas id="canvas" width="300" height="300"></canvas>
  <div>
    <label for="speed">Speed:</label>
    <input type="number" id="speed" value="0.09" step="0.001">
    <button id="reset">Reset</button>
  </div>

  <script>
    const canvas = document.getElementById('canvas');
```

```
const centerY = canvas.height / 2;
const numDots = 50;
const dotSize = width * 0.075; // 7.5% of the total width
let speed = parseFloat(document.getElementById('speed').value);
const dots = Array.from({ length: numDots }, (_, i) => ({
  angle: (i / numDots) * 2 * Math.PI,
  distance: width / 2,
}));
const dotPositions = [];

const updateSpeed = () => {
  speed = parseFloat(document.getElementById('speed').value);
};

const resetAnimation = () => {
  speed = 0.09;
  document.getElementById('speed').value = speed;
};

const drawDot = (dot, i) => {
  const x = centerX + dot.distance * Math.cos(dot.angle);
  const y = centerY + dot.distance * Math.sin(dot.angle);
  ctx.beginPath();
  ctx.arc(x, y, dotSize / 2, 0, 2 * Math.PI);
  ctx.fillStyle = `rgba(0, 0, 255, ${1 - i / numDots})`;
  ctx.fill();
};

const updateDot = (dot, i) => {
  dot.angle += speed;
  dot.distance += 0.001;
  if (dot.angle >= 2 * Math.PI) dot.angle -= 2 * Math.PI;
  if (dot.distance > width / 2 * 0.84) dot.distance -= width * 0.16;

  // Check for overlaps and adjust direction
  const x = centerX + dot.distance * Math.cos(dot.angle);
  const y = centerY + dot.distance * Math.sin(dot.angle);
  dots.forEach((otherDot, j) => {
    if (i !== j) {
      const otherX = centerX + otherDot.distance * Math.cos(otherDot.angle);
      const otherY = centerY + otherDot.distance * Math.sin(otherDot.angle);
      const distance = Math.sqrt((x - otherX) ** 2 + (y - otherY) ** 2);
      if (distance < dotSize) {
        if (x < centerX) {
          dot.angle -= 0.01;
        } else {
```

```
            dot.distance -= 0.01;
          } else {
            dot.distance += 0.01;
          }
        }
      }
    });
  };

  const animate = () => {
    ctx.clearRect(0, 0, canvas.width, canvas.height);
    ctx.beginPath();
    ctx.arc(centerX, centerY, width / 2, 0, 2 * Math.PI);
    ctx.stroke();

    dots.forEach((dot, i) => {
      drawDot(dot, i);
      updateDot(dot, i);
    });

    // Gather positions every 60ms
    if (dotPositions.length === 0 || performance.now() - dotPositions[dotPositions.length - 1].t
      dotPositions.push({
        time: performance.now(),
        positions: dots.map(dot => ({
          x: centerX + dot.distance * Math.cos(dot.angle),
          y: centerY + dot.distance * Math.sin(dot.angle)
        }))
      });
    }

    requestAnimationFrame(animate);
  };

  document.getElementById('speed').addEventListener('input', updateSpeed);
  document.getElementById('reset').addEventListener('click', resetAnimation);

  animate();
</script>
</body>
</html>
```