Arc Drawing Sample Application Documentation

Project Overview

This is a sample application that demonstrates how to implement a 3-point arc drawing tool on an HTML canvas. The application is built using vanilla JavaScript with an object-oriented approach, utilizing the HTML5 Canvas API for rendering. This sample could be used as a reference for implementing similar drawing functionality in RxCore or other canvas-based applications.

Architecture

The project follows a modular architecture with clear separation of concerns:

1. **Main Application** (`main.js`): Entry point that initializes the canvas and tools.
2. **Utility Modules**:
    - `utils.js`: General utility functions
    - `Tool.js`: Base class for all drawing tools
    - `TempTools.js`: Manages drawing tools and their interactions
    - `RenderUtils.js`: Handles all rendering operations

Component Details

Canvas Setup (`main.js`)

The application initializes by setting up a full-window canvas that automatically resizes when the browser window changes:

```
const canvas = document.getElementById("canvas");
const setupCanvas = () => {
  // Initialize canvas size
  resizeCanvas(canvas);
  // Add resize listener to adjust canvas when window size changes
  window.addEventListener("resize", resizeCanvas);
};
setupCanvas();
```

Tool Management System (`TempTools.js`)

The `TempTools` class serves as a manager for all drawing tools and handles:

- Setting the active tool
- Managing event listeners for tools
- Adding completed drawings to the collection
- Rendering all drawn items
- Providing snapping points for precision drawing

```
export default class TempTools {
  constructor(canvas) {
    this.canvas = canvas;
    this.context = canvas.getContext("2d");
    this.currentTool = null;
    this.items = []; // Array to store drawn items
    // ...
  }

  // Methods for managing tools and drawing
  setTool(tool) {
    /* ... */
  }
  clear() {
    /* ... */
  }
  addTool(tool) {
    /* ... */
  }
  drawAll() {
    /* ... */
  }
  getSnappingPoint(coordinate) {
    /* ... */
  }
}
```

Base Tool Class (`Tool.js`)

`Tool` serves as the foundation for all drawing tools, providing basic properties and a unique ID system:

```
export default class Tool {
  constructor(canvas) {
    this.canvas = canvas;
    this.context = canvas.getContext("2d");
    this.id = ids++; // Auto-incrementing ID
  }
}
```

Arc Drawing Tool (`ArcTool3Point` in `main.js`)

The primary drawing tool in the application is the `ArcTool3Point`, which creates arcs by specifying three points:

- First point: Starting point of the arc
- Second point: Another point on the arc
- Third point: Ending point of the arc

The tool handles the following interactions:

- Mouse down: Places points in sequence
- Mouse move: Shows a preview of the arc
- Completion: Adds the arc to the collection when all three points are placed

```
class ArcTool3Point extends Tool {
  constructor(canvas) {
    super(canvas);
    this.startX = null;
    this.startY = null;
    this.secondX = null;
    this.secondY = null;
    this.endX = null;
    this.endY = null;
    this.isDrawing = false;
    this.customType = "arc3point";
  }

  // Methods for handling user interactions
  onMouseDown(event) {
    /* ... */
  }
  onMouseMove(event) {
    /* ... */
  }
  onMouseUp(event) {
    /* ... */
  }
  redraw() {
    /* ... */
  }
}
```

Rendering Utilities (`RenderUtils.js`)

`RenderUtils` is a static class containing methods for drawing various elements:

- Points
- Initial arcs (based on two points)
- Complete arcs (based on three points)
- Labels with surrounding boxes
- Hollow circles (for snapping indicators)

The most complex functionality is the `calculateCircumCircle` method, which calculates the center and radius of a circle based on three points using geometric formulas.

User Interaction Flow

1. User clicks the "Add Arc" button
2. An instance of `ArcTool3Point` is created and set as the active tool
3. User clicks to place the first point
4. User clicks to place the second point
    - During mouse movement, a preview arc is shown
5. User clicks to place the third point
    - The arc is added to the collection
    - The tool is reset

Technical Implementation: Creating Arcs

Arc Creation Process

The arc drawing is implemented using a mathematical concept called the circumcircle - a unique circle that passes through three given points. Here's how it's accomplished:

1. Collecting Three Points

- When the user initiates the arc tool and clicks three times on the canvas, each click stores a point:
    - First click: `startX`, `startY` (starting point of the arc)
    - Second click: `secondX`, `secondY` (a point on the arc)
    - Third click: `endX`, `endY` (ending point of the arc)

2. Mathematical Foundation

The `calculateCircumCircle` method in `RenderUtils.js` computes the center and radius of a circle that passes through all three points:

```
static calculateCircumCircle = (p1, p2, p3) => {
  // Calculate determinant coefficients
  const A = x1 * (y2 - y3) - y1 * (x2 - x3) + x2 * y3 - x3 * y2;

  // Check for collinear points (A ≈ 0)
  if (Math.abs(A) < 1e-10) {
    // Fallback to a simpler arc when points are collinear
    // ...
  }

  // Calculate center coordinates and radius using standard formulas
  const centerX = -B / (2 * A);
  const centerY = -C / (2 * A);
  const radius = Math.sqrt((B * B + C * C - 4 * A * D) / (4 * A * A));

  return { center: { x: centerX, y: centerY }, radius };
}
```

3. Direction Determination

The `drawArc` method determines whether to draw the arc clockwise or counterclockwise based on the position of the middle point relative to the start and end points:

```
// Check angles between points to determine arc direction
const diff12 = (a2 - a1 + 2 * Math.PI) % (2 * Math.PI);
const diff13 = (a3 - a1 + 2 * Math.PI) % (2 * Math.PI);
const diff32 = (a2 - a3 + 2 * Math.PI) % (2 * Math.PI);

// Determine clockwise or counterclockwise direction
let anticlockwise = false;
if (diff12 < Math.PI) {
  if (diff13 > diff12 || diff32 > diff12) {
    anticlockwise = true;
  }
} else {
  if (diff13 < diff12 && diff32 < diff12) {
    anticlockwise = false;
  } else {
    anticlockwise = true;
  }
}
```

4. Rendering

The arc is drawn using the HTML5 Canvas `arc()` method:

```
ctx.beginPath();
ctx.strokeStyle = color;
ctx.lineWidth = 2;
ctx.arc(center.x, center.y, radius, a1, a2, anticlockwise);
ctx.stroke();
```

5. Edge Case Handling

For cases where the three points are nearly collinear (making a proper circle impossible), a fallback mechanism creates an arc based on just two points with an artificially inflated radius:

```
// When points are collinear, use a fallback approach
const distance = Math.hypot(p2.x - p1.x, p2.y - p1.y);
const radius = 1.2 * distance; // 20% larger than the distance between points
```