Data Science Team ~ Sh. Munesh Dutt

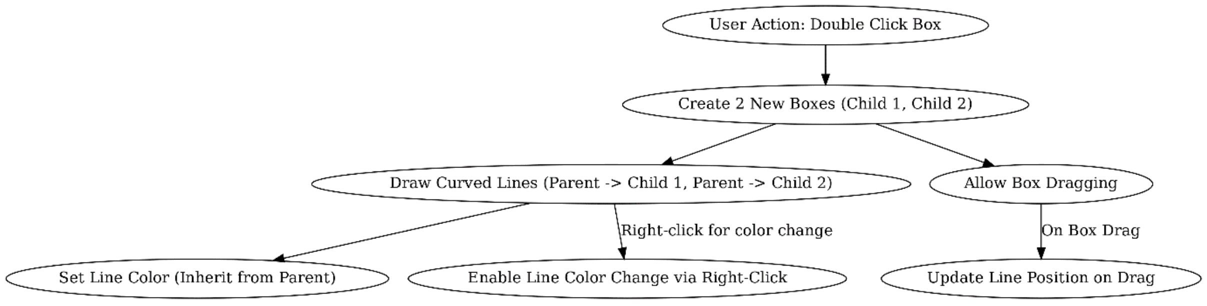
Project Documentation 08 Sep 2024, Sunday

Documentation for Dynamic Box-Connection System

Overview

This JavaScript code implements a dynamic, interactive system where users can manipulate boxes on a canvas, connect them with curved lines, zoom in/out, and pan around the canvas. Boxes are connected with lines (curved paths), and new boxes can be created dynamically, inheriting certain attributes like connection color from their parent boxes. Users can also modify the color of the lines, and newly created connections inherit the color from their parent.

Diagram


User Action: Double Click Box
Create 2 New Boxes (Child 1, Child 2)
Draw Curved Lines (Parent -> Child 1, Parent -> Child 2)
Allow Box Dragging
Right-click for color change
On Box Drag
Set Line Color (Inherit from Parent)
Enable Line Color Change via Right-Click
Update Line Position on Drag

Key Features

1. **Zoom and Pan Support**: Users can zoom in and out of the canvas using the mouse wheel and pan around by clicking and dragging.
2. **Box Creation**: New boxes can be created dynamically by double-clicking on existing boxes. These new boxes are automatically connected to the original box with curved lines.
3. **Drag and Drop**: All boxes can be dragged around the canvas, and the connections (lines) between boxes update in real-time to maintain the visual linkage between boxes.

4. **Curved Connections**: The boxes are connected by SVG curved lines, which adjust dynamically to maintain the connection as boxes are moved.
5. **Line Color Customization**: Right-clicking on a connection (line) allows users to change the color of that line. The color is inherited by new connections formed when additional boxes are created.
6. **Hierarchy Management**: The system supports a hierarchical structure of boxes where newly created boxes are considered "children" of the box that spawned them. The visibility of these sub-boxes can be toggled (hidden or shown).

Detailed Feature Description

1. Zoom and Pan

The system allows users to zoom in/out and pan the view of the canvas.

- **Zoom**: Mouse wheel controls zooming. The zoom() function adjusts the scale of the entire container by a zoomSensitivity factor. Zooming is centered at the cursor position, and the view is adjusted accordingly to maintain the relative position.
- **Pan**: By clicking and dragging, the entire container can be moved (panned). The panning feature works by adjusting panX and panY, which represent the translation of the container. These transformations are applied via CSS using transform: translate() and scale().

```
document.addEventListener('wheel', function (e) {
  e.preventDefault();
  zoom(e.deltaY, e);  // Zoom using mouse wheel
});
```

2. Box Creation

Users can double-click on any existing box to spawn two new child boxes connected to the original box. These connections inherit the line color from the parent box's connection.

- **Double-Click**: When a box is double-clicked, two new boxes (child boxes) are created, positioned to the left and right of the parent box.
- **Curved Connection**: Each new child box is connected to the parent via curved SVG paths, which are dynamically created and updated.
- **Hierarchy**: The parent-child relationship between boxes is stored in a boxHierarchy structure, which is used to hide/show the child boxes when the parent box is toggled.

```
box.addEventListener('dblclick', function () {
  // Create new boxes and connections
  const newBox1 = createBox('purple', newBox1Top,
newBox1Left);
  const newBox2 = createBox('orange', newBox2Top,
newBox2Left);
  ...
});
```

3. Drag and Drop

All boxes on the canvas can be moved by dragging. When a box is dragged, the connections (lines) that link it to other boxes are dynamically updated to maintain the visual linkage.

- **Mouse Events**: mousedown, mousemove, and mouseup events are used to implement the dragging feature.
- **Connection Update**: As the box moves, the positions of the connections (lines) are recalculated using the createOrUpdateCurvedLink() function, ensuring the lines between boxes remain attached.

```
box.addEventListener('mousedown', function (e) {
  // Handle dragging logic
  ...
});

document.addEventListener('mousemove', function (e) {
  if (isDragging) {
    // Update box position and the attached connections
    createOrUpdateCurvedLink(connection.box1, connection.box2,
connection.path);
  }
});
```

4. Curved Connections

Boxes are connected with curved lines, represented by SVG paths. These lines update dynamically when the boxes are moved.

- **Center Points**: The lines are drawn between the center points of the two boxes.
- **Control Point**: The control point for the curve is computed as the midpoint between the two boxes on the X-axis, while being adjusted vertically to create a smooth curve.

```
function createOrUpdateCurvedLink(box1, box2, path) {
  const box1Rect = box1.getBoundingClientRect();
  const box2Rect = box2.getBoundingClientRect();
  const box1CenterX = box1Rect.left + box1Rect.width / 2;
  const box1CenterY = box1Rect.top + box1Rect.height / 2;
  const box2CenterX = box2Rect.left + box2Rect.width / 2;
  const box2CenterY = box2Rect.top + box2Rect.height / 2;

  const controlPointX = (box1CenterX + box2CenterX) / 2;
  const controlPointY = Math.min(box1CenterY,
box2CenterY) + 300;
```

```
  const pathData = `M ${box1CenterX} ${box1CenterY} Q
${controlPointX} ${controlPointY}, ${box2CenterX}
${box2CenterY}`;
  path.setAttribute("d", pathData);
}
```

5. Line Color Customization

Right-clicking on any connection (line) allows users to change its color. A prompt is displayed where the user can enter a new color (e.g., "red" or "#FF0000").

- **Context Menu**: The contextmenu event is used to detect right-clicks on lines. The prompt() function is used to get user input for the new color.
- **Color Inheritance**: Newly created connections inherit the color of their parent box's connection.

```
function enableLineColorChange(path) {
  path.addEventListener('contextmenu', function (e) {
    e.preventDefault();
    const newColor = prompt("Enter a new color for the line
(e.g., red, #FF0000):", path.getAttribute("stroke"));
    if (newColor) {
      path.setAttribute("stroke", newColor);
    }
  });
}
```

6. Hierarchy and Visibility Management

Boxes can have sub-boxes (children) that are dynamically created. The visibility of these sub-boxes can be toggled by double-clicking on the parent box. When hidden, the child boxes and their connections disappear, and when shown, they reappear.

- **Recursive Toggling**: The hideSubBoxes() and showSubBoxes() functions recursively hide or show all descendant boxes and their connections.

```
function hideSubBoxes(box) {
  const subBoxes = boxHierarchy.get(box);
  if (subBoxes) {
    subBoxes.forEach(subBox => {
      subBox.element.style.display = 'none';
      hideSubBoxes(subBox.element);
    });}}
```

How the Features are Connected

Connection Between Boxes and Lines

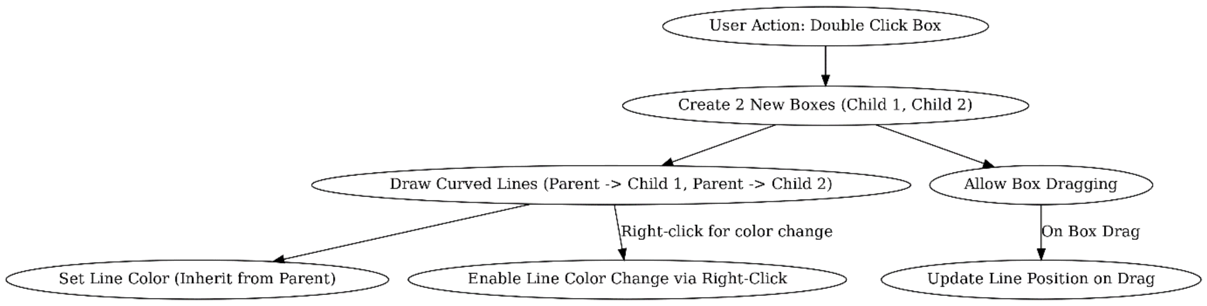
The primary interaction in the system is between boxes and their connections (lines). Each box is linked to other boxes through SVG curved lines, and every time a box is moved, its connections are recalculated and redrawn.

Line Color Propagation

Whenever a line's color is manually changed, this color is inherited by future connections that stem from the same box. This inheritance allows users to create visually distinct sections of their box hierarchy by assigning unique colors.

Flowchart and Diagram

Below is a simplified diagram of the system:


User Action: Double Click Box
Create 2 New Boxes (Child 1, Child 2)
Draw Curved Lines (Parent -> Child 1, Parent -> Child 2)
Allow Box Dragging
Right-click for color change
On Box Drag
Set Line Color (Inherit from Parent)
Enable Line Color Change via Right-Click
Update Line Position on Drag

```
[ Box 1 ]

      |

      | [ Connection Line ]

      |

  [ Box 2 ] ------- [ Box 3 ]

      |

      | [ Connection Line ]

      |

 [ Sub-Box 1 ] --- [ Sub-Box 2 ]
```

1. **Boxes**: Represented by [Box X], where each box can be created by double-clicking on an existing box.
2. **Connection Lines**: Represented by | [Connection Line] |, connecting the center of one box to another.
3. **Sub-Boxes**: When you double-click on any box, two child boxes ([Sub-Box X]) are created, inheriting the parent's line color.

Hierarchy Example

- **Parent Box (Box 1)**: When double-clicked, creates two new child boxes (Box 2 and Box 3).
- **Child Boxes (Box 2, Box 3)**: They, in turn, can be double-clicked to create their own sub-boxes.
- **Curved Lines**: SVG paths link parent and child boxes and update dynamically as the boxes move.

Conclusion

This dynamic box-connection system provides an intuitive way to manage hierarchical relationships between objects with visual feedback in the form of draggable boxes and adjustable connections. Through features like line color inheritance, zoom/pan support, and recursive hiding/showing of child boxes, users can easily create complex visual structures.

4o