Uge 38: Maze Generation

En gennemgang af forskellige algoritmer

https://weblog.jamisbuck.org/2011/2/7/maze-generation-algorithm-recap

Vi ser på visualisering af forskellige labyrint-genererings-algoritmer

Datamodellering: Labyrinter

Diskussion: Hvordan repræsenterer man en 2D labyrint med celler og vægge?

Recursive Backtracking / Growing Tree

Vi fokuserer på to algoritmer.

Recursive Backtracking (pseudokode)

```
generateStep(grid) {
    // Get unvisited neighbors
    // If there are unvisited neighbors:
    //   - pick a random one of them
    //   - carve a hole through the wall
    //   - push current cell on stack
    //   - make that neighbor the current cell
    // If not, make the top of stack the current cell
    // If still not, you're done
}

generateMaze(grid) {
    // Initialize grid
    // Set the current cell as start
    // While there are unvisited cells:
    //   - generateStep(grid)
    //   - Redraw the maze
}
```

Growing Tree Algorithm

```
generateStep(grid) {
    // Pick a random cell from the list
    // Pick a random neighbor to that cell
    // Carve a hole through the wall
    // Add both cells back to the list if they have unvisited neighbors
    // If not, you're done
}

generateMaze(grid) {
    // Initialize grid
    // Set the current cell as start
    // While there are unvisited cells:
    //   - generateStep(grid)
    //   - Redraw the maze
}
```

HTML Canvas Tutorial

For at tegne labyrinter skal vi bruge HTML Canvas.

Her er en gennemgang af de grundlæggende elementer:

HTML-skabelon

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Canvas Tutorial</title>
</head>
<body>
    <canvas id="myCanvas" width="800" height="800"></canvas>
    <script>
        // JavaScript kode kommer her
    </script>
</body>
</html>
```

Canvas context og tegne streger og firkanter

```javascript
// Få fat i canvas elementet
const canvas = document.getElementById('myCanvas');
const ctx = canvas.getContext('2d');

// Tegn en firkant
ctx.fillStyle = '#ff6b6b';
ctx.fillRect(50, 50, 100, 100);

// Tegn en kant
ctx.strokeStyle = '#000';
ctx.lineWidth = 2;
ctx.strokeRect(50, 50, 100, 100);
```

`ctx` (context) er det objekt vi bruger til at tegne på canvas.

Det indeholder alle tegne-metoderne som:

- `fillRect(x, y, width, height)` - tegner en fyldt firkant
- `strokeRect(x, y, width, height)` - tegner en kant omkring en firkant
- `beginPath()`, `moveTo()`, `lineTo()`, `stroke()` - til at tegne linjer

Funktion til at Tegne et Grid

```
function drawGrid(ctx, cellSize, cols, rows) {
    ctx.strokeStyle = '#333';
    ctx.lineWidth = 1;

    // Tegn vertikale linjer
    for (let x = 0; x <= cols; x++) {
        ctx.beginPath();
        ctx.moveTo(x * cellSize, 0);
        ctx.lineTo(x * cellSize, rows * cellSize);
        ctx.stroke();
    }

    // Tegn horisontale linjer
    for (let y = 0; y <= rows; y++) {
        ctx.beginPath();
        ctx.moveTo(0, y * cellSize);
        ctx.lineTo(cols * cellSize, y * cellSize);
        ctx.stroke();
    }
}

// Kald funktionen
const cellSize = 20;
const cols = 20;
const rows = 20;
drawGrid(ctx, cellSize, cols, rows);
```

Objekt-orienteret labyrint

```
class Cell {
    constructor(x, y) {
        this.x = x;
        this.y = y;
        this.walls = { top: true, right: true, bottom: true, left: true };
        this.visited = false;
    }

    draw(ctx, cellSize) {
        const x = this.x * cellSize;
        const y = this.y * cellSize;

        ctx.strokeStyle = '#000';
        ctx.lineWidth = 2;

        if (this.walls.top) {
            ctx.beginPath();
            ctx.moveTo(x, y);
            ctx.lineTo(x + cellSize, y);
            ctx.stroke();
        }
        // ... tegn de andre vægge
    }
}
```