ISCG6420 Semester 1 2024

Exercise: Canvas Animation

These instructions will take you through the process of creating an animation using JavaScript to be displayed on an HTML canvas.

New Webpage

Open a folder in VSCode. Inside the folder, create a new html file and add the boilerplate HTML code (html:5).

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Canvas Animation Exercise</title>
</head>
<body>
</body>
</html>
```

Add a canvas element inside the body, and add a script under it pointing to an external js file.

```
</head>
<body>
    <canvas id="myCanvas" width="800" height="600"></canvas>
    <script src="./canvas.js"></script>
</body>
</html>
```

Add a style element to the head with the following canvas style:

```
    <style>
        canvas {
            border: #000 solid 1px;
            background-color: #1b1b1b;
        }
    </style>
</head>
```

Save the html file.

JavaScript

Create a new file called “canvas.js”. Inside the file, establish a 2D context for the canvas:

```
const canvas = document.getElementById("myCanvas");
const ctx = canvas.getContext("2d");
```

Create a function object called Logo. Inside set the following properties and methods:

```
function Logo() {
    this.loading = true;
    this.image = new Image();
    this.image.src = "./dvd-logo.png";
    this.image.onload = () => { this.loading = false; };
    this.scaledWidth = this.image.width / 4;
    this.scaledHeight = this.image.height / 4;
    this.x = random(0, canvas.clientWidth - this.scaledWidth);
    this.y = random(0, canvas.clientHeight - this.scaledHeight);
    this.velocityX = 1;
    this.velocityY = 1;

    this.update = () => {
        this.x += this.velocityX;
        this.y += this.velocityY;
    }

    this.draw = (context) => {
        context.drawImage(
            this.image,
            this.x,
            this.y,
            this.scaledWidth,
            this.scaledHeight
        );
    }
}
```

- The Logo object contains an image and variables for displaying the image on the canvas.
- Loading is a boolean that tells us when the image has finished loading. The onload event sets Loading to false when it is safe to draw the image to the canvas.
- ScaledWidth and Height are the size the image will be scaled down to when drawn to the canvas. In this case it will be 1 quarter its original size.
- X and Y are the image position coordinates. They are set to a random number between 0 and the furthest they can move before leaving the canvas.
- VelocityX and Y are the amount of movement applied each time the position is updated (this is sometimes called the ‘step’ size.
- The update function applies the movement step to the position.

- The draw function calls drawImage on the given context (which will be the canvas) and uses the Logo variables to draw it in the right place at the right size.

Create an empty array for storing our animated elements (logos). Add a click event listener to the canvas with a function that creates a new Logo object and pushes it to the array. Call run(), which will start the animation process (we will make the functions to do this next).

```
let logos = [];

canvas.addEventListener("click", () => {logos.push(new Logo());});

logos.push(new Logo());

run();
```

Create 5 functions: run, update, draw, random, and checkWallCollision.

```
function run() {…
}

function update() {…
}

function draw() {…
}

function random(min, max) {…
}

function checkWallCollision(object) {…
}
```

Run

Run is a simple function. Call update, draw then request to be called again. This is a pseudo-recursive function, because it calls itself, albeit via the window object.

```
function run() {
    update();
    draw();
    window.requestAnimationFrame(run);
}
```

Update

Update loops through the logos array, and for each object, calls its update method, and passes it to the checkWallCollisions function for further processing.

```
function update() {
    for (let i = 0; i < logos.length; i++) {
        logos[i].update();
        checkWallCollision(logos[i]);
    }

}
```

Draw

Draw clears the canvas with clearRect, then loops through the logos array, and for each object, calls its draw method.

```
function draw() {
    ctx.clearRect(0, 0, canvas.clientWidth, canvas.clientHeight);

    for (let i = 0; i < logos.length; i++) {
        if(!logos[i].loading) {
            logos[i].draw(ctx);
        }
    }
}
```

Random

Random uses a bit of math trickery to get a random number between the min and max values provided. JS Math.random() only provides a value between 0 and 1, so we multiply it by max – min and then add min back. This will scale up the value and offset by min so we don't overmultiply and leave the intended range.

```
function random(min, max) {
    return Math.floor((Math.random() * (max - min + 1)) + min);
}
```

Wall collision handling

We want the logos to hit a wall and bounce off. To achieve this, we first need to identify the conditions that depict the logo trying to leave the canvas bounds:

```
function checkWallCollision(object) {
    if (object.x + object.scaledWidth >= canvas.clientWidth) {
        // right wall collision
    }
    else if (object.x <= 0) {
        // left wall collision
    }

    if (object.y + object.scaledHeight >= canvas.clientHeight) {
        // bottom wall collision
    }
    else if (object.y <= 0) {
        // top wall collision
    }
}
```

For each collision, we want to reverse the direction of velocity in the axis that the collision occurred in. EG: hitting the left or right wall should reverse velocityX. velocityX = -velocityX.

```
function checkWallCollision(object) {
    // right wall collision
    if (object.x + object.scaledWidth >= canvas.clientWidth) {
        object.velocityX = -object.velocityX;
    }
    // left wall collision
    else if (object.x <= 0) {
        object.velocityX = -object.velocityX;
    }

    // bottom wall collision
    if (object.y + object.scaledHeight >= canvas.clientHeight) {
        object.velocityY = -object.velocityY;
    }
    // top wall collision
    else if (object.y <= 0) {
        object.velocityY = -object.velocityY;
    }
}
```

Test

Save the js file and open the html file in your web browser, or start a server with the Live Server plugin.

Upon load, one logo should appear and animate diagonally and bouncing off the canvas bounds. Clicking on the canvas should add additional logos at random locations.

Next Steps

- Set the spawn position to the location of the mouse click.
- Add audio to the website to play a sound each time a collision occurs.
- Detect when any two logos collide and implement a response.

Exercise complete.