**Development of the hover effect over the years**

The concept of hover interaction has its roots in the developments of graphical user interfaces and the evolution of input devices. There are several instances that can connect to the birth of the hover effect. One of them is the invention of a mouse by Douglas Engelbart in the 1960s. The mouse allowed users to **move** a cursor on the screen and as a result, designers began exploring ways to provide feedback when the cursor was positioned over an element. Many other factors like Apple’s Macintosh from 1984 which features a mouse and graphical elements, with Microsoft Windows in the 1990s, users could hover over desktop icons, buttons, and other UI elements to receive visual feedback. Then with the development of HTML, CSS, and JavaScript in the 1990s-2000s and the CSS hover pseudo-class, the hover interaction became mainstream. It is, however, still in the beginning phases of being applied to all touchscreen devices.

**Brainstorming:**

With the context from the timeline of the hover effect, I believe it is strongly linked with the movement of the mouse.

As the mouse allowed users to *move* a cursor, I wanted to relate this idea of movement of the mouse with what is defined as hover in today's time.

“Hover is an interactive visual change that occurs when a user places their cursor over a specific element on a webpage without clicking on it.”

I looked into various hover effects that are widely used on the webpages. I went over some examples like the hamburger menu hover interactions, highlighting/glowing a card, zooming in on something when hovered over, and more.

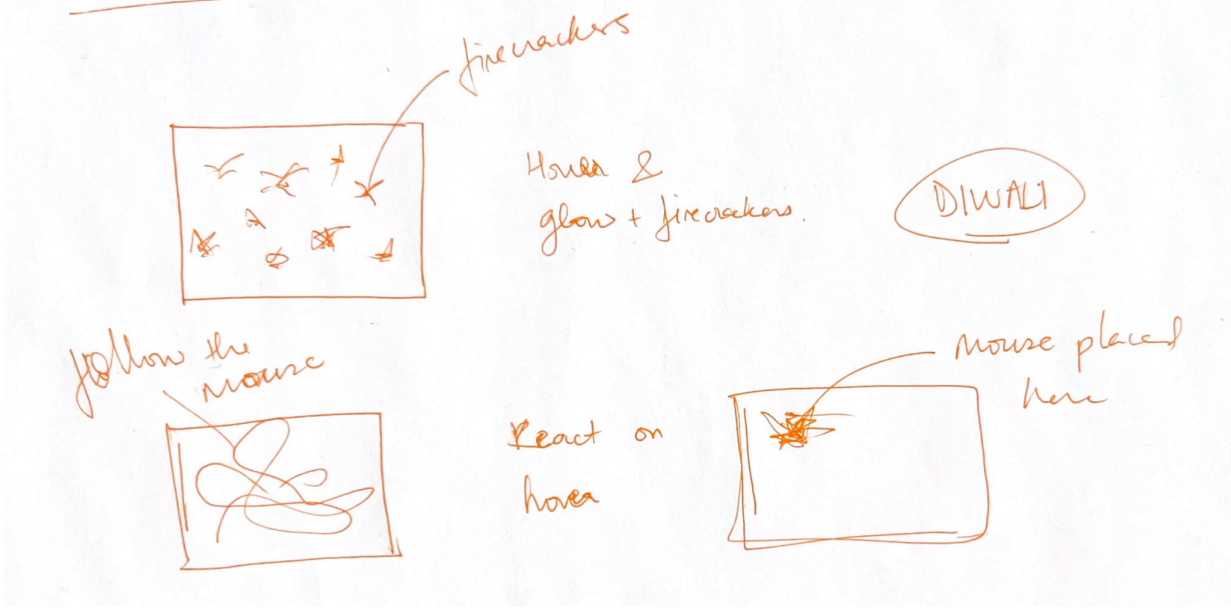
I asked myself – what if I make the entire screen as an “element” that is to be hovered over to interact with it?

This question led me down a rabbit hole to find mouse trail effects in JavaScript.

**Concept**

This project fell during the time of a festival I celebrate back home, Diwali. Since I am an international student and was not going to be able to celebrate it this year, I thought of “celebrating” Diwali through this exercise. For context, during Diwali we burst firecrackers and fuljhadi (sparklers) (although I have not done them in a while as I believe they are a harm to the environment, I did enjoy them in my childhood and it is still a major part of celebration of this festival in India).

I wanted to create a resemblance of firecrackers but not firecrackers directly. I wanted an iteration/ variation of the firecrackers with the beautiful colors seen during fireworks at night. After a few rough sketches, I decided to have balls follow the mouse only when the mouse is hovered on the screen and no balls appear when there is no movement.



Code

To tie my ideas together, I looked up some videos on YouTube and found the following video which I started my assignment.

<https://www.youtube.com/watch?v=aEptSB3fbqM>

This code introduced to me something called Canvas 2d, context, “this.”, “mouse.”, “this.speedX”, hsl, “context.arc”, and more.

//canvas 2d rendering

const canvas = document.getElementById('canvas'); //get canvas elements from html

const context = canvas.getContext('2d'); //2d rendering context of canvas

//canvas width and height is set to full Screen

canvas.width = window.innerWidth;

canvas.height = window.innerHeight;

//array to store instances of the ball class

let ballsArray = [];

//ball class constructor

function Ball(x, y) {

    this.x = x; // x coordinate assigned to the object ball created

    this.y = y;

    this.radius = 20; // base radius of the ball

    this.color = 'hsl(' + hue + ', 100%, 50%)'; //hue, saturation

    //random initial speed for the ball

    this.speedX = Math.random() \* 3 - 1.5;

    this.speedY = Math.random() \* 3 - 1.5;

    //ball properties

    this.update = () => {

        //position of the ball is updated by adding speed of the ball x 5

        if (this.radius >= 10) {

            this.x += this.speedX \* 5; //this.x =  this.x + this.speedX \* 5;

            this.y += this.speedY \* 5;

        }

        if (this.radius <= 9) {

            this.x += this.speedX \* 2;

            this.y += this.speedY \* 2;

        }

         //to keep reducing the radius of the ball

        if (this.radius > 4) {

            this.radius -= 0.9;

        }

        if (this.radius < 4) {

            this.radius -= 0.2;

        }

    };

    //to render the ball on canvas

    this.draw = () => {

        context.fillStyle = this.color;

        context.beginPath();

        context.arc(this.x, this.y, this.radius, 0, Math.PI \* 2);

        context.fill();

    };

}

//function to render all balls in the ballsArray

function renderBalls() {

    for (let i = 0; i < ballsArray.length; i++) {

        ballsArray[i].draw();

        ballsArray[i].update();

        //remove balls with radius equal to or less than 0.1

        if (ballsArray[i].radius <= 0.1) {

            ballsArray.splice(i, 1);

            i--;

        }

    }

}

let mouseX = 0;

let mouseY = 0;

let hue = 0;

//this eventlistener adds new instances of ball to ballsArrary when the mouse is moved

canvas.addEventListener('mousemove', (e) => {

    mouseX = e.clientX;

    mouseY = e.clientY;

    for (let i = 0; i < 5; i++) {

        ballsArray.push(new Ball(mouseX, mouseY));

    }

});

//funciton to update and render the canvas continously

function animate() {

    context.clearRect(0, 0, canvas.width, canvas.height); //clear canvas

    renderBalls(); //calling the renderballs function

    hue++; //to create variation of color (hue)

    requestAnimationFrame(animate);

}

animate();

I created a canvas for rendering the balls on the screen. I created an empty array as it is dynamic, it allows continuous creation, rendering, and removal of balls during the animation loop.

A screen shot of a computer code

Description automatically generated

The Ball class consists of the properties and behaviors

Of a ball in the animation. Instances of this class are created and added to the ‘ballsArray’ in response to mouse movements and they are continuously animated and updated in the ‘renderballs’ function and overall animation loop.

A purple dots on a black background

Description automatically generated

The constructor function ‘Ball’ is used for creating objects that represent balls. The initial radius is set to 20 and I have used hsl (hue, saturation, and lightness) which gives the perfect colors I was looking for, took this reference from the video linked above.

Since I wanted to move the balls around the screen to represent the “fireworks”, I had to set the random values generated to be positive and negative in the x and y directions. Also, the larger ball sizes move faster than the smaller ones and all balls also keep reducing in size.

A screenshot of a computer program

Description automatically generated

This function is the animation loop which iterates over each ball in the array, draws it on the canvas, updates its properties, and removes balls that have become very small.

A screen shot of a computer code

Description automatically generated

To determine the position of the mouse pointer, clientX, a property by DOM is used to locate the coordinates. With each mouse movement, 5 balls are added to the array.

A computer screen with text on it

Description automatically generated

This function animation clears the canvas, calls renderBalls function, and adds a variation of color by incrementing hue by 1.

A screen shot of a computer code

Description automatically generated