Processamento de imagem

Nomes:	RGMs:
Eduarda Fernandes	29204356
Davi Santos de Andrade	31075550
Johnatan Caetano	30087155
Everman	30333717
Daniel Medeiros	29381169

- Objetivos:
    - Implementação de filtros que retirem um ou mais canais de cor de uma imagem dependendo do sistema de cores que esteja sendo aplicado.
    - Implementação do filtro Blur, mais qualquer outro filtro convencional.
    - Implementação de uma interface para execução das etapas em qualquer ordem podendo aproveitar o resultado de uma operação anterior para executar um novo processamento de imagem.

- Equipamentos/ferramentas utilizados:
  Para desenvolver esta aplicação conforme descrito anteriormente, utilizamos um notebook Samsung Core i5 de décima geração como plataforma principal. Optamos pela IDE chamada Visual Studio Code devido à sua interface intuitiva e fácil manipulação. A linguagem de programação escolhida foi Javascript, devido à sua versatilidade e eficiência. Para garantir uma organização adequada do projeto, decidimos utilizar o GitHub para armazenar e colaborar no código, permitindo que todos os membros do grupo pudessem contribuir de forma eficiente.

- Procedimento experimental:

Para cumprir o primeiro objetivo descrito anteriormente, começamos selecionando elementos do DOM (Document Object Model) e definindo algumas variáveis e arrays para manipular esses elementos posteriormente.

```
const inputFile = document.querySelector("#picture__input");
const picture = document.querySelector(".picture");
const pictureImage = document.querySelector(".picture__image");
const pictureImageTxt = "Choose an image";
const convertToExcelBtn = document.getElementById("convertToExcel");
const modifyColorsBtn = document.getElementById("modifyColors");
const convertToCMYKBtn = document.getElementById("convertToCMYK");
const convertToGrayScaleBtn = document.getElementById("convertToGrayScale");
const grayScaleSelector = document.getElementById("grayScaleSelector");
const increaseColorContrastBtn = document.getElementById("increaseColorContrast");
const increaseGrayContrastBtn = document.getElementById("increaseGrayContrast");
let appliedFilters = [];
const noRedBtn = document.getElementById("noRed");
const noGreenBtn = document.getElementById("noGreen");
const noBlueBtn = document.getElementById("noBlue");
const blurBtn = document.getElementById("Blur");
const resetBtn = document.getElementById("Reset");
```

Na segunda parte do código, desenvolvemos um objeto CORES, que mapeia identificadores de cores (números de 1 a 20) para suas respectivas informações, incluindo o nome da cor e seu valor RGB (Red, Green, Blue). O objeto CORES pode ser utilizado para diversas finalidades, como atribuir cores específicas a elementos de uma interface, aplicar filtros de cor, ou realizar outras operações relacionadas a cores em um contexto de manipulação de imagens.

```
const CORES = {
    "1": {"nome": "Vermelho", "valor": [255, 0, 0]},
    "2": {"nome": "Verde", "valor": [0, 255, 0]},
    "3": {"nome": "Azul", "valor": [0, 0, 255]},
    "4": {"nome": "Amarelo", "valor": [255, 255, 0]},
    "5": {"nome": "Magenta", "valor": [255, 0, 255]},
    "6": {"nome": "Ciano", "valor": [0, 255, 255]},
    "7": {"nome": "Marrom", "valor": [128, 0, 0]},
    "8": {"nome": "Verde Escuro", "valor": [0, 128, 0]},
    "9": {"nome": "Azul Escuro", "valor": [0, 0, 128]},
    "10": {"nome": "Oliva", "valor": [128, 128, 0]},
    "11": {"nome": "Roxo", "valor": [128, 0, 128]},
    "12": {"nome": "Teal", "valor": [0, 128, 128]},
    "13": {"nome": "Cinza", "valor": [128, 128, 128]},
    "14": {"nome": "Prata", "valor": [192, 192, 192]},
    "15": {"nome": "Branco", "valor": [255, 255, 255]},
    "16": {"nome": "Preto", "valor": [0, 0, 0]},
    "17": {"nome": "Laranja", "valor": [255, 165, 0]},
    "18": {"nome": "Rosa", "valor": [255, 192, 203]},
    "19": {"nome": "Turquesa", "valor": [64, 224, 208]},
    "20": {"nome": "Lavanda", "valor": [230, 230, 250]}
};
```

No terceiro trecho do código, elaboramos duas funções: enableButtons e disableButtons, que são responsáveis por habilitar e desabilitar um conjunto de botões na interface do usuário.

```
function enableButtons() {
    convertToExcelBtn.removeAttribute("disabled");
    modifyColorsBtn.removeAttribute("disabled");
    convertToCMYKBtn.removeAttribute("disabled");
    convertToGrayScaleBtn.removeAttribute("disabled");
    increaseColorContrastBtn.removeAttribute("disabled");
    increaseGrayContrastBtn.removeAttribute("disabled");
    noRedBtn.removeAttribute("disabled");
    noGreenBtn.removeAttribute("disabled");
    noBlueBtn.removeAttribute("disabled");
    blurBtn.removeAttribute("disabled");
    resetBtn.removeAttribute("disabled");
}

// Função para desabilitar os botões
function disableButtons() {
    convertToExcelBtn.setAttribute("disabled", true);
    modifyColorsBtn.setAttribute("disabled", true);
    convertToCMYKBtn.setAttribute("disabled", true);
    convertToGrayScaleBtn.setAttribute("disabled", true);
    increaseColorContrastBtn.setAttribute("disabled", true);
    increaseGrayContrastBtn.setAttribute("disabled", true);
    noRedBtn.setAttribute("disabled", true);      Davi140903, há 23 horas • last commit
    noGreenBtn.setAttribute("disabled", true);
    noBlueBtn.setAttribute("disabled", true);
    blurBtn.setAttribute("disabled", true);
    resetBtn.setAttribute("disabled", true);
}
```

Logo após no quarto trecho do código adicionamos ouvintes de eventos (event listeners) para manipular o comportamento de arrastar e soltar (drag and drop) uma imagem sobre um elemento da interface do usuário identificado pela classe picture.

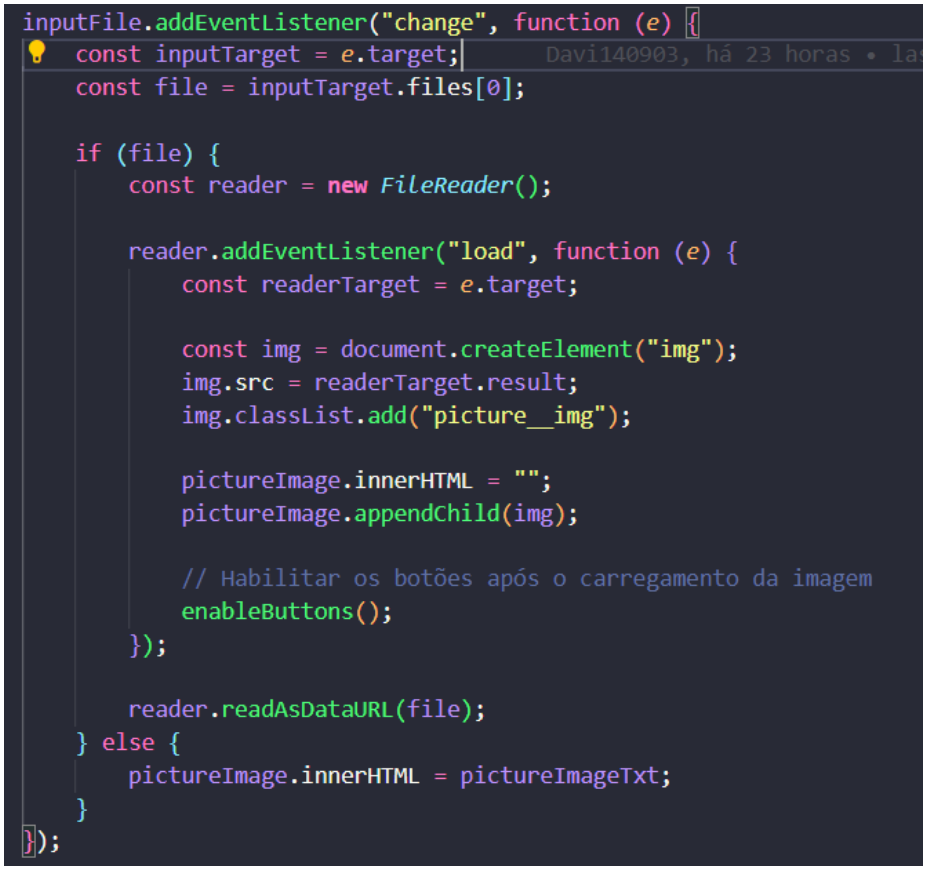
```
picture.addEventListener("dragover", function (e) {
    e.preventDefault();
    picture.classList.add("dragover");
});

picture.addEventListener("dragleave", function (e) {
    e.preventDefault();
    picture.classList.remove("dragover");
});
```

O qui
imag e
exibi ual
ao usuário (adicionando e removendo a classe dragover), manipula o

conteúdo do elemento pictureImage, e habilita botões de ação que permitem modificar a imagem carregada. Se nenhum arquivo for solto, o texto padrão é exibido na área da imagem:

```
picture.addEventListener("drop", function (e) {
    e.preventDefault();
    picture.classList.remove("dragover");
    const file = e.dataTransfer.files[0];

    if (file) {
        const reader = new FileReader();

        reader.addEventListener("load", function (e) {
            const readerTarget = e.target;

            const img = document.createElement("img");
            img.src = readerTarget.result;
            img.classList.add("picture__img");

            pictureImage.innerHTML = "";
            pictureImage.appendChild(img);

            // Habilitar os botões após o carregamento da imagem
            enableButtons();
        });

        reader.readAsDataURL(file);
    } else {
        pictureImage.innerHTML = pictureImageTxt;
    }
});
```

Logo após, o código trata o evento de mudança no campo de entrada de arquivo inputFile, que ocorre quando o usuário seleciona um arquivo para carregar. Quando um arquivo é selecionado, ele é lido e exibido na página. Durante o processo, o código verifica se há um arquivo selecionado, manipula o conteúdo do elemento pictureImage, e habilita botões de ação que permitem modificar a imagem carregada. Se nenhum arquivo for selecionado, o texto padrão é exibido na área da imagem:



```
inputFile.addEventListener("change", function (e) {
    const inputTarget = e.target;      Davi140903, há 23 horas • la
    const file = inputTarget.files[0];

    if (file) {
        const reader = new FileReader();

        reader.addEventListener("load", function (e) {
            const readerTarget = e.target;

            const img = document.createElement("img");
            img.src = readerTarget.result;
            img.classList.add("picture__img");

            pictureImage.innerHTML = "";
            pictureImage.appendChild(img);

            // Habilitar os botões após o carregamento da imagem
            enableButtons();
        });

        reader.readAsDataURL(file);
    } else {
        pictureImage.innerHTML = pictureImageTxt;
    }
});
```

Na próxima etapa, o código permite que o usuário converta uma imagem carregada na página em um arquivo Excel (.xlsx). Quando o botão convertToExcelBtn é clicado, a imagem é desenhada em um canvas, os dados da imagem são extraídos e convertidos em uma matriz, essa matriz é transformada em uma planilha do Excel usando a biblioteca SheetJS (XLSX), e finalmente um arquivo Excel é gerado e disponibilizado para download. Se nenhuma imagem for carregada, um alerta é exibido pedindo ao usuário para carregar uma imagem antes de continuar:

```
convertToExcelBtn.addEventListener("click", function() {
    const imgElement = document.querySelector(".picture__img");
    if (!imgElement) {
        alert("Por favor, carregue uma imagem primeiro.");
        return;
    }

    const canvas = document.createElement("canvas");
    const ctx = canvas.getContext("2d");
    const imgWidth = imgElement.width;
    const imgHeight = imgElement.height;
    canvas.width = imgWidth;
    canvas.height = imgHeight;
    ctx.drawImage(imgElement, 0, 0, imgWidth, imgHeight);
    const imgData = ctx.getImageData(0, 0, imgWidth, imgHeight).data;

    const matrix = [];
    for (let y = 0; y < imgHeight; y++) {
        const row = [];
        for (let x = 0; x < imgWidth; x++) {
            const idx = (y * imgWidth + x) * 4;
            const r = imgData[idx];
            const g = imgData[idx + 1];
            const b = imgData[idx + 2];
            row.push(`(${r}, ${g}, ${b})`); // Armazenar o valor RGB de cada pixel
        }
        matrix.push(row);
    }

    const wb = XLSX.utils.book_new();
    const ws = XLSX.utils.aoa_to_sheet(matrix);
    XLSX.utils.book_append_sheet(wb, ws, "Imagem");
    const excelData = XLSX.write(wb, { bookType: "xlsx", type: "binary" });
    const blob = new Blob([s2ab(excelData)], { type: "application/octet-stream" });
    const url = URL.createObjectURL(blob);
    const a = document.createElement("a");
    a.href = url;
    a.download = "imagem.xlsx";
    document.body.appendChild(a);
    a.click();
    URL.revokeObjectURL(url);
});
```

Essa próxima função é útil ao trabalhar com operações que envolvem manipulação de arquivos binários em JavaScript, como a criação de arquivos Excel, manipulação de imagens, entre outros. Ao receber uma string como entrada, ela converte essa string em um ArrayBuffer contendo os bytes correspondentes aos caracteres da string. Isso é frequentemente usado em conjunto com APIs que manipulam dados

binários, onde é necessário representar os dados de uma forma específica para serem processados corretamente.

```
function s2ab(s) {
    const buf = new ArrayBuffer(s.length);
    const view = new Uint8Array(buf);
    for (let i = 0; i < s.length; i++) view[i] = s.charCodeAt(i) & 0xff;
    return buf;
}
```

Na próxima etapa, criamos um código que permite que o usuário modifique as cores de uma imagem carregada na página. Quando o botão modifyColorsBtn é clicado, o usuário pode escolher uma nova cor e todas as áreas da imagem que têm a mesma cor que um pixel aleatório selecionado anteriormente serão substituídas pela nova cor selecionada. Este processo é realizado no canvas, e a imagem modificada é exibida ao usuário. Se nenhuma imagem estiver carregada, um alerta é exibido pedindo ao usuário para carregar uma imagem antes de continuar.

```
modifyColorsBtn.addEventListener("click", function() {
    // Verificar se há uma imagem carregada
    const imgElement = document.querySelector(".picture__img");
    if (!imgElement) {
        alert("Por favor, carregue uma imagem primeiro.");
        return;
    }

    // Selecionar um pixel aleatório
    const canvas = document.createElement("canvas");
    const ctx = canvas.getContext("2d");
    const imgWidth = imgElement.width;
    const imgHeight = imgElement.height;
    canvas.width = imgWidth;
    canvas.height = imgHeight;
    ctx.drawImage(imgElement, 0, 0, imgWidth, imgHeight);      Davi140903, há 24 horas • last commit
    const imgData = ctx.getImageData(0, 0, imgWidth, imgHeight).data;
    const randomX = Math.floor(Math.random() * imgWidth);
    const randomY = Math.floor(Math.random() * imgHeight);
    const pixelIndex = (randomY * imgWidth + randomX) * 4;
    const selectedColor = {
        r: imgData[pixelIndex],
        g: imgData[pixelIndex + 1],
        b: imgData[pixelIndex + 2]
    };

    // Criar um seletor de cores para o usuário escolher a nova cor
    const newColor = prompt("Escolha uma nova cor:\n" +
                            Object.keys(CORES).map(key => `${key}: ${CORES[key].nome}`).join("\n"));
    if (!CORES[newColor]) {
        alert("Cor inválida.");
        return;
    }

    // Obter a nova cor selecionada pelo usuário
    const [newR, newG, newB] = CORES[newColor].valor;
```

```
    // Substituir todos os pixels com a mesma cor pelo novo valor
    for (let i = 0; i < imgData.length; i += 4) {
        if (imgData[i] === selectedColor.r &&
            imgData[i + 1] === selectedColor.g &&
            imgData[i + 2] === selectedColor.b) {
            imgData[i] = newR;
            imgData[i + 1] = newG;
            imgData[i + 2] = newB;
        }
    }

    // Desenhar a imagem modificada no canvas
    ctx.putImageData(new ImageData(imgData, imgWidth, imgHeight), 0, 0);

    // Exibir a imagem modificada no elemento .picture__img
    const modifiedImg = new Image();
    modifiedImg.src = canvas.toDataURL();
    modifiedImg.classList.add("picture__img");

    // Substituir a imagem original pela imagem modificada
    const pictureImage = document.querySelector(".picture__image");
    pictureImage.innerHTML = "";
    pictureImage.appendChild(modifiedImg);
});
```

Logo após elaboramos uma função para pegar a cor de cada pixel da imagem e converter de RGB para CMYK.

```
function rgbToCmyk(r, g, b) {
    // Normalizar os valores RGB para o intervalo [0, 1]
    const R = r / 255;
    const G = g / 255;
    const B = b / 255;

    // Calcular os valores CMY
    const C = 1 - R;
    const M = 1 - G;
    const Y = 1 - B;

    // Encontrar o valor mínimo entre CMY
    const K = Math.min(C, Math.min(M, Y));

    // Se K for 1, retornar CMYK como (0, 0, 0, 1) para preto puro
    if (K === 1) return [0, 0, 0, 1];

    // Calcular os valores CMYK finais
    const CMYK = [
        (C - K) / (1 - K),
        (M - K) / (1 - K),
        (Y - K) / (1 - K),
        K
    ];

    // Retornar os valores CMYK
    return CMYK.map(value => Math.round(value * 100)); // Multiplicar por 100 para obter valores de 0 a 100
}
```

Esse próximo código permite que o usuário converta uma imagem carregada para o formato CMYK e faça o download da imagem resultante. Quando o botão convertToCMYKBtn é clicado, a imagem é convertida de RGB para CMYK usando a função rgbToCmyk, e a imagem CMYK resultante é disponibilizada para download como um arquivo PNG. Se nenhuma imagem estiver carregada, um alerta é exibido pedindo ao usuário para carregar uma imagem antes de continuar.

```
convertToCMYKBtn.addEventListener("click", function() {
    // Verificar se há uma imagem carregada
    const imgElement = document.querySelector(".picture__img");
    if (!imgElement) {
        alert("Por favor, carregue uma imagem primeiro.");
        return;
    }

    // Criar um canvas para manipular a imagem
    const canvas = document.createElement("canvas");
    const ctx = canvas.getContext("2d");

    // Configurar o canvas com as dimensões da imagem
    const imgWidth = imgElement.width;
    const imgHeight = imgElement.height;
    canvas.width = imgWidth;
    canvas.height = imgHeight;

    // Desenhar a imagem no canvas
    ctx.drawImage(imgElement, 0, 0, imgWidth, imgHeight);

    // Obter os dados da imagem no formato de matriz de pixels
    const imgData = ctx.getImageData(0, 0, imgWidth, imgHeight);

    // Converter os pixels da imagem de RGB para CMYK
    for (let i = 0; i < imgData.data.length; i += 4) {
        const r = imgData.data[i];
        const g = imgData.data[i + 1];
        const b = imgData.data[i + 2];
        const cmyk = rgbToCmyk(r, g, b);

        // Atualizar os valores de cor do pixel para CMYK
        imgData.data[i] = cmyk[0]; // C
        imgData.data[i + 1] = cmyk[1]; // M
        imgData.data[i + 2] = cmyk[2]; // Y
        // Ignorar o canal K (preto) para manter o visual RGB na imagem resultante
    }

    // Desenhar a imagem modificada no canvas
    ctx.putImageData(imgData, 0, 0);

    // Criar um link para fazer o download da imagem CMYK      Davi140903, há 24 horas •
    const a = document.createElement("a");
    a.href = canvas.toDataURL();
    a.download = "imagem_cmyk.png";
    document.body.appendChild(a);
    a.click();

    // Remover o link após o download
    document.body.removeChild(a);
});
```

Nas próximas sessões, definimos as funções que convertem a imagem em escala de cinza, seguindo os conceitos apresentados em sala:

- Luminosidade:

```
function luminosityGrayScale(imgData) {
    for (let i = 0; i < imgData.data.length; i += 4) {
        const gray = 0.21 * imgData.data[i] + 0.72 * imgData.data[i + 1] + 0.07 * imgData.data[i + 2];
        imgData.data[i] = gray;
        imgData.data[i + 1] = gray;
        imgData.data[i + 2] = gray;
    }
    return imgData;
}
```

- Média ponderada:

```
function averageGrayScale(imgData) {
    for (let i = 0; i < imgData.data.length; i += 4) {
        const gray = (imgData.data[i] + imgData.data[i + 1] + imgData.data[i + 2]) / 3;
        imgData.data[i] = gray;
        imgData.data[i + 1] = gray;
        imgData.data[i + 2] = gray;
    }
    return imgData;
}
```

- Dessaturação:

```
function desaturationGrayScale(imgData) {
    for (let i = 0; i < imgData.data.length; i += 4) {      Davi140903, há 24 horas • last comm
        const gray = (Math.max(imgData.data[i], imgData.data[i + 1], imgData.data[i + 2]) +
            Math.min(imgData.data[i], imgData.data[i + 1], imgData.data[i + 2])) / 2;
        imgData.data[i] = gray;
        imgData.data[i + 1] = gray;
        imgData.data[i + 2] = gray;
    }
    return imgData;
}
```

- Valor máximo:

```
function maximumGrayScale(imgData) {
    for (let i = 0; i < imgData.data.length; i += 4) {
        const gray = Math.max(imgData.data[i], imgData.data[i + 1], imgData.data[i + 2]);
        imgData.data[i] = gray;
        imgData.data[i + 1] = gray;
        imgData.data[i + 2] = gray;
    }
    return imgData;
}
```

- Valor mínimo:

```
function minimumGrayScale(imgData) {
    for (let i = 0; i < imgData.data.length; i += 4) {
        const gray = Math.min(imgData.data[i], imgData.data[i + 1], imgData.data[i + 2]);
        imgData.data[i] = gray;
        imgData.data[i + 1] = gray;
        imgData.data[i + 2] = gray;
    }
    return imgData;
}
```

Após essa definição, foi desenvolvido um método para o usuário selecionar qual o tipo de conversão deseja:

```
function convertToGrayScale(imgData, method) {
    switch (method) {
        case "1":
            return averageGrayScale(imgData);
        case "2":
            return luminosityGrayScale(imgData);
        case "3":
            return desaturationGrayScale(imgData);
        case "4":
            return maximumGrayScale(imgData);
        case "5":
            return minimumGrayScale(imgData);
        default:
            return averageGrayScale(imgData); // Média ponderada como padrão
    }
}
```

```
convertToGrayScaleBtn.addEventListener("click", function() {
    const imgElement = document.querySelector(".picture__img");
    if (!imgElement) {
        alert("Por favor, carregue uma imagem primeiro.");
        return;
    }

    const canvas = document.createElement("canvas");
    const ctx = canvas.getContext("2d");
    const imgWidth = imgElement.width;
    const imgHeight = imgElement.height;
    canvas.width = imgWidth;
    canvas.height = imgHeight;
    ctx.drawImage(imgElement, 0, 0, imgWidth, imgHeight);
    let imgData = ctx.getImageData(0, 0, imgWidth, imgHeight);

    const method = prompt("Escolha o método de conversão para escala de cinza:\n" +
                          "1. Média ponderada\n" +
                          "2. Luminosidade\n" +
                          "3. Dessaturação\n" +
                          "4. Máximo\n" +
                          "5. Mínimo\n" +
                          "Digite o número correspondente:");
    if (!["1", "2", "3", "4", "5"].includes(method)) {
        alert("Método inválido.");
        return;
    }

    imgData = convertToGrayScale(imgData, method);

    ctx.putImageData(imgData, 0, 0);

    const a = document.createElement("a");
    a.href = canvas.toDataURL();
    a.download = `imagem_grayscale_${method}.png`;
    document.body.appendChild(a);
    a.click();
    document.body.removeChild(a);
});
```

- Análise de resultados:

A próxima função é útil quando é necessário aumentar o contraste das cores em uma imagem. Ela opera diretamente nos dados de imagem (na forma de um objeto ImageData) e aumenta o contraste aplicando uma transformação aos valores RGB de cada pixel. O fator de contraste determina a intensidade do aumento do contraste. O resultado é uma imagem com cores mais vívidas e distintas.

```
function increaseColorContrast(imgData, factor) {
    for (let i = 0; i < imgData.data.length; i += 4) {
        // Obter os componentes de cor RGB do pixel atual
        const r = imgData.data[i];
        const g = imgData.data[i + 1];
        const b = imgData.data[i + 2];

        // Calcular o valor médio dos componentes de cor RGB
        const avg = (r + g + b) / 3;

        // Calcular os novos valores dos componentes de cor usando o fator de contraste
        const newR = avg + factor * (r - avg);
        const newG = avg + factor * (g - avg);
        const newB = avg + factor * (b - avg);

        // Definir os novos valores dos componentes de cor para o pixel atual
        imgData.data[i] = clamp(newR, 0, 255);
        imgData.data[i + 1] = clamp(newG, 0, 255);
        imgData.data[i + 2] = clamp(newB, 0, 255);
    }
    return imgData;
}
```

Este próximo trecho de código permite ao usuário aumentar o contraste das cores em uma imagem carregada, solicitando um fator de contraste e aplicando-o à imagem. A imagem resultante, com contraste aumentado, é disponibilizada para download como um arquivo PNG. Se nenhuma imagem estiver carregada, um alerta é exibido pedindo ao usuário para carregar uma imagem antes de continuar.

```
function clamp(value, min, max) {
    return Math.min(Math.max(value, min), max);
}

increaseColorContrastBtn.addEventListener("click", function() {
    const imgElement = document.querySelector(".picture__img");
    if (!imgElement) {
        alert("Por favor, carregue uma imagem primeiro.");
        return;
    }

    const canvas = document.createElement("canvas");
    const ctx = canvas.getContext("2d");
    const imgWidth = imgElement.width;
    const imgHeight = imgElement.height;
    canvas.width = imgWidth;
    canvas.height = imgHeight;
    ctx.drawImage(imgElement, 0, 0, imgWidth, imgHeight);
    let imgData = ctx.getImageData(0, 0, imgWidth, imgHeight);

    // Solicitar ao usuário que insira o fator de contraste
    const factor = parseFloat(prompt("Insira o fator de contraste (por exemplo, 1.5):"));
    if (isNaN(factor)) {
        alert("Fator de contraste inválido.");
        return;
    }

    // Aumentar o contraste da imagem usando o fator especificado
    imgData = increaseColorContrast(imgData, factor);

    ctx.putImageData(imgData, 0, 0);

    const a = document.createElement("a");
    a.href = canvas.toDataURL();
    a.download = `imagem_contrast_${factor}.png`;
    document.body.appendChild(a);
    a.click();
    document.body.removeChild(a);
});
```

Na próxima etapa foi definido um método que aumenta o contraste nos tons de cinza:

```
function increaseGrayContrast(imgData, factor) {
    for (let i = 0; i < imgData.data.length; i += 4) {
        // Obter o valor de cinza do pixel atual
        const gray = imgData.data[i];

        // Ajustar o valor de cinza para aumentar o contraste
        const newGray = (gray - 128) * factor + 128;

        // Definir o novo valor de cinza para o pixel atual
        imgData.data[i] = clamp(newGray, 0, 255);
        imgData.data[i + 1] = clamp(newGray, 0, 255);
        imgData.data[i + 2] = clamp(newGray, 0, 255);
    }
    return imgData;
}

increaseGrayContrastBtn.addEventListener("click", function() {
    const imgElement = document.querySelector(".picture__img");
    if (!imgElement) {
        alert("Por favor, carregue uma imagem primeiro.");
        return;
    }

    const canvas = document.createElement("canvas");
    const ctx = canvas.getContext("2d");
    const imgWidth = imgElement.width;
    const imgHeight = imgElement.height;
    canvas.width = imgWidth;
    canvas.height = imgHeight;
    ctx.drawImage(imgElement, 0, 0, imgWidth, imgHeight);
    let imgData = ctx.getImageData(0, 0, imgWidth, imgHeight);

    // Solicitar ao usuário que insira o fator de contraste
    const factor = parseFloat(prompt("Insira o fator de contraste (por exemplo, 1.5):"));
    if (isNaN(factor)) {
        alert("Fator de contraste inválido.");
        return;
    }
      Davi140903, ontem • last commit
    // Aumentar o contraste da imagem em tons de cinza usando o fator especificado
    imgData = increaseGrayContrast(imgData, factor);

    ctx.putImageData(imgData, 0, 0);

    const a = document.createElement("a");
    a.href = canvas.toDataURL();
    a.download = `imagem_gray_contrast_${factor}.png`;
    document.body.appendChild(a);
    a.click();
    document.body.removeChild(a);
});
```

Na próxima etapa foi definido um método que aplica um filtro de remoção de cor na imagem:

```
function applyFilter(color) {
    // Verificar se há uma imagem carregada
    const imgElement = document.querySelector(".picture__img");
    if (!imgElement) {
        alert("Por favor, carregue uma imagem primeiro.");
        return;
    }

    // Adiciona o filtro à lista de filtros aplicados
    appliedFilters.push(color);

    // Criar um canvas para manipular a imagem
    const canvas = document.createElement("canvas");
    const ctx = canvas.getContext("2d");

    // Configurar o canvas com as dimensões da imagem
    const imgWidth = imgElement.width;
    const imgHeight = imgElement.height;
    canvas.width = imgWidth;
    canvas.height = imgHeight;

    // Desenhar a imagem no canvas
    ctx.drawImage(imgElement, 0, 0, imgWidth, imgHeight);

    // Obter os dados da imagem no formato de matriz de pixels
    let imageData = ctx.getImageData(0, 0, imgWidth, imgHeight);

    // Aplicar cada filtro acumulado à imagem
    appliedFilters.forEach(filter => {
        for (let i = 0; i < imageData.data.length; i += 4) {
            if (filter === 'red') imageData.data[i] = 0;       // Remove o canal vermelho
            if (filter === 'green') imageData.data[i + 1] = 0; // Remove o canal verde
            if (filter === 'blue') imageData.data[i + 2] = 0;  // Remove o canal azul
        }
    });

    // Desenhar a imagem filtrada no canvas
    ctx.putImageData(imageData, 0, 0);

    // Exibir a imagem modificada no elemento .picture__img
    const modifiedImg = new Image();
    modifiedImg.src = canvas.toDataURL();
    modifiedImg.classList.add("picture__img");

    // Substituir a imagem original pela imagem modificada
    const pictureImage = document.querySelector(".picture__image");
    pictureImage.innerHTML = "";
    pictureImage.appendChild(modifiedImg);
}
```

Logo em seguida foi definido as ações dos botões que referenciam as cores que serão retiradas:

```
// Adicionar evento para o botão Vermelho
noRedBtn.addEventListener("click", function() {
    applyFilter('red');
});

// Adicionar evento para o botão Verde
noGreenBtn.addEventListener("click", function() {
    applyFilter('green');
});

// Adicionar evento para o botão Azul
noBlueBtn.addEventListener("click", function() {
    applyFilter('blue');
});
```

Logo após, a lógica que traz o filtro Blur a imagem que traz o efeito de desfoque:

```
// Função para aplicar efeito de desfoque (blur)
function applyBlurEffect() {
    // Verificar se há uma imagem carregada
    const imgElement = document.querySelector(".picture__img");
    if (!imgElement) {
        alert("Por favor, carregue uma imagem primeiro.");
        return;
    }

    // Criar um canvas para manipular a imagem
    const canvas = document.createElement("canvas");
    const ctx = canvas.getContext("2d");

    // Configurar o canvas com as dimensões da imagem
    const imgWidth = imgElement.width;
    const imgHeight = imgElement.height;
    canvas.width = imgWidth;
    canvas.height = imgHeight;

    // Desenhar a imagem no canvas
    ctx.drawImage(imgElement, 0, 0, imgWidth, imgHeight);

    // Aplicar o efeito de desfoque (blur) à imagem no canvas
    ctx.filter = "blur(5px)"; // Ajuste o valor de desfoque conforme necessário
    ctx.drawImage(canvas, 0, 0, imgWidth, imgHeight, 0, 0, imgWidth, imgHeight);

    // Exibir a imagem modificada no elemento .picture__img
    const modifiedImg = new Image();
    modifiedImg.src = canvas.toDataURL();
    modifiedImg.classList.add("picture__img");

    // Substituir a imagem original pela imagem modificada
    const pictureImage = document.querySelector(".picture__image");
    pictureImage.innerHTML = "";
    pictureImage.appendChild(modifiedImg);
}
```

```
// Variável para armazenar a imagem original
let originalImg = null;

// Função para carregar a imagem
function loadImage(file) {
    const reader = new FileReader();

    reader.addEventListener("load", function (e) {
        const readerTarget = e.target;

        originalImg = document.createElement("img");
        originalImg.src = readerTarget.result;
        originalImg.classList.add("picture__img");

        const pictureImage = document.querySelector(".picture__image");
        pictureImage.innerHTML = "";
        pictureImage.appendChild(originalImg);
    });

    reader.readAsDataURL(file);
}

// Adicione um evento ao input de arquivo para carregar a imagem
inputFile.addEventListener("change", function (e) {
    const inputTarget = e.target;
    const file = inputTarget.files[0];
    if (file) {
        loadImage(file);
    }
});
```

- Análise de resultados:
  A análise do resultado mostra que existem diversas formas de se manipular uma imagem, seja alterar a cor, o contraste, ou adicionar algum filtro, e através deste trabalho foi possível identificar toda a lógica e ver na prática todo esse processo acontecendo através da programação, e após a implementação foi possível identificar como a imagem é manipulada, e mudada desde a cor, até mesmo a forma de se enxergar a mesma.

- Considerações finais:
  Uma nova etapa foi integrada ao projeto. Agora, conseguimos reunir todo o conhecimento adquirido ao longo do curso, além de implementar uma interface para facilitar o uso pelo usuário do sistema. É uma etapa interessante, pois todo o aprendizado foi colocado a prova, e com certeza todo o grupo absorveu bastante

informação nova e todos poderão ver as imagens com outros olhos.