JOGO DA MEMÓRIA

Criar arquivo inicial:

Index.html

Usando o bootstrap:

Getbootstrap.com

Lembrando que pelo emet você pode fazer:

div.row.container-fluid <tab>

e

button.btn.btn-success#jogar <tab> e button.btn.btn-danger#mostrar\_Tudo <tab>

Resultado meu:

    <div class="row container-fluid">

    </div>

Acrescente entre as tags body:

<body class="container-fluid">

Na tag <body> inclua uma classe bootstrap:

Lembre-se: Este código deve ficar entre as tags <meta> do html

Clique em downloads

Copie e cole os links baseados no CDN:

    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.2/dist/css/bootstrap.min.css" rel="stylesheet"

        integrity="sha384-Zenh87qX5JnK2Jl0vWa8Ck2rdkQ2Bzep5IDxbcnCeuOxjzrPF/et3URy9Bv1WTRi" crossorigin="anonymous">

    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.2/dist/js/bootstrap.bundle.min.js"

        integrity="sha384-OERcA2EqjJCMA+/3y+gxIOqMEjwtxJY7qPCqsdltbNJuaOe923+mo//f6V8Qbsw3"

        crossorigin="anonymous"></script>

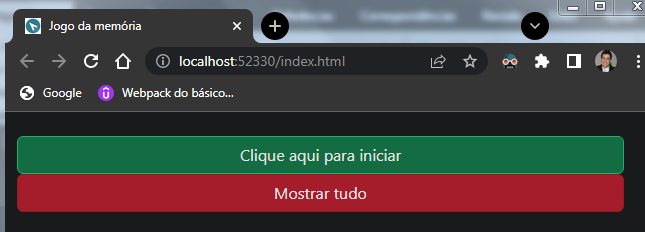
<br/>

<div class="row container-fluid">

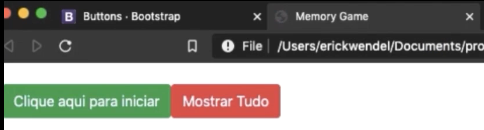
    <button type="button" class="btn btn-success" id="jogar">Clique aqui para iniciar</button>

    <button type="button" class="btn btn-danger" id="mostrar\_Tudo">Mostrar tudo</button>

</div>



Resultado do professor:



Neste caso, é bem provável que o bootstrap (na versão que uso) tenha alterado a aparência padrão das class usadas aqui

Observe: Aqui vc terá que preencher

...

<div class="card" style="width: 18rem;">

    <img src="..." class="card-img-top" alt="...">

    <div class="card-body">

        <p class="card-text">Some quick example text to build on the card title and make up the bulk of the card's content.</p>

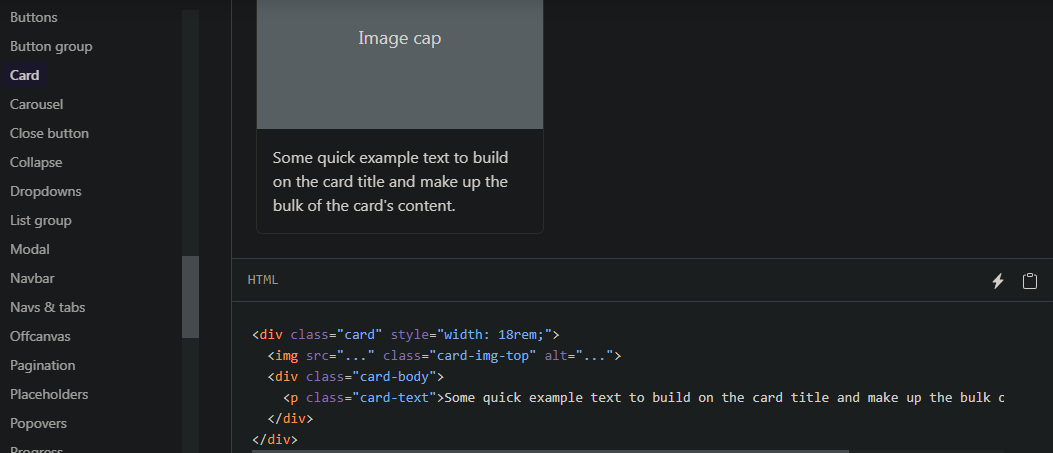
    </div>

</div>

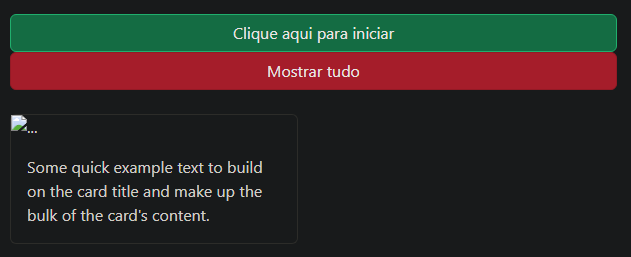
...

Copie o código html e cole em seu código

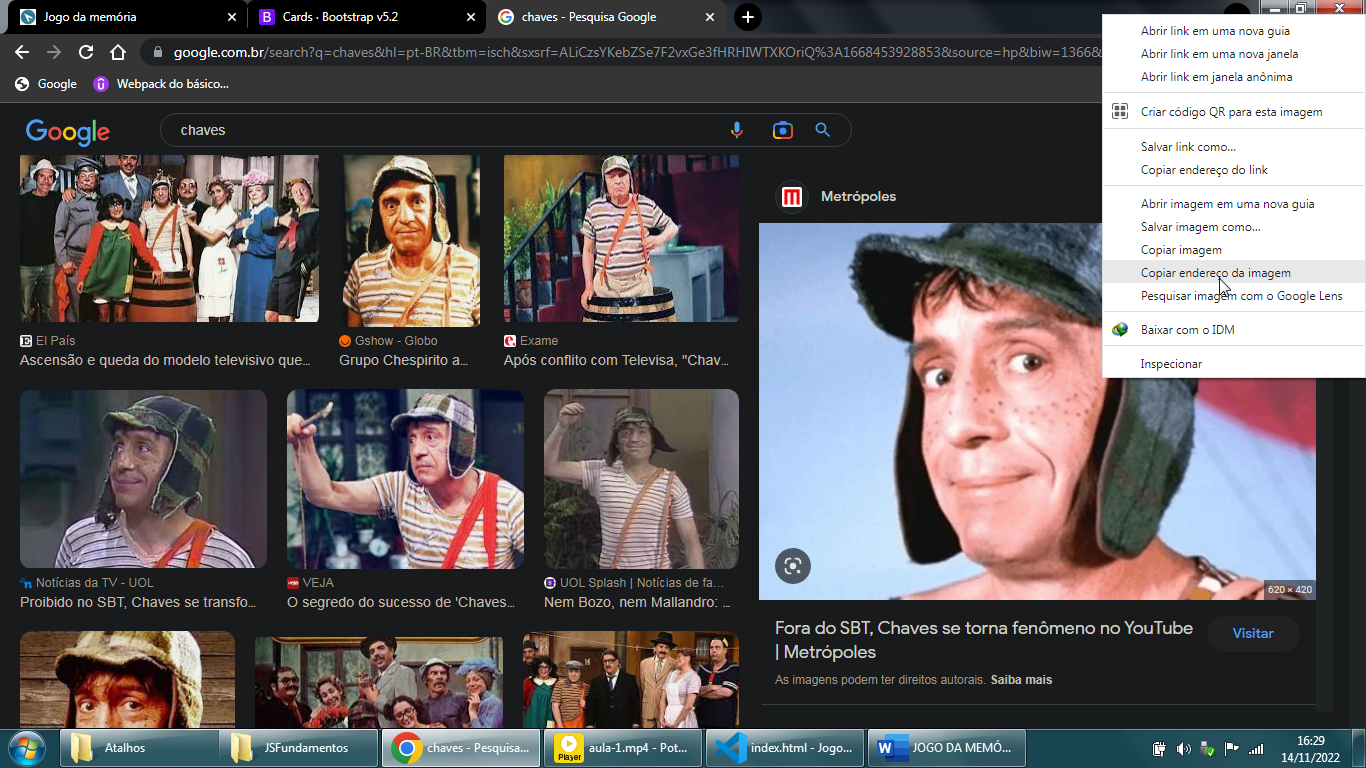
Na documentação do bootstrap -> card – Selecione algo do tipo:

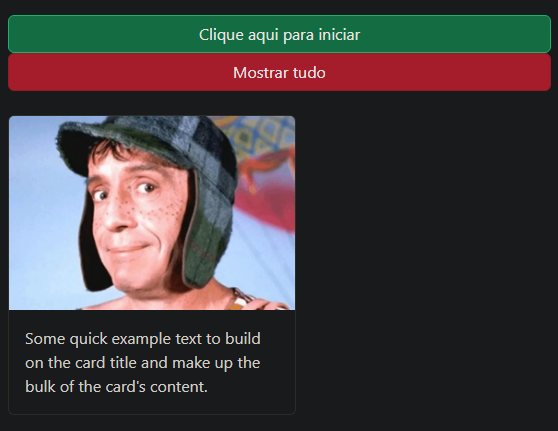


Resultado:



Para fazer um simples teste, procure por chaves no google e depois em copiar endereço da imagem clicando com o botão direito e acrescente no src





Procure ícones que serão usados no jogo. Precisaremos de uma imagem para o verso do card e 4 ‘heróis’

Você pode usar o site: Iconfinder.com

Crie um diretório ‘arquivos’ no seu projeto e salve as imagens

Neste código:

...

<div class="card" style="width: 18rem;">

    <img src="..." class="card-img-top" alt="...">

    <div class="card-body">

        <p class="card-text">Some quick example text to build on the card title and make up the bulk of the card's content.</p>

    </div>

</div>

...

Resultado:

...

<div class="card" style="width: 18rem;">

    <img src="..." class="card-img-top" alt="...">

</div>

...

Recorte este código e em seguida, usando o emet:

div.row>div.col-md-3\*4 <tab>

<div class="row">

    <div class="col-md-3"></div>

    <div class="col-md-3"></div>

    <div class="col-md-3"></div>

    <div class="col-md-3"></div>

</div>

E cole o código (recortado anteriormente) dentro de cada uma das div class ‘col-md-3’. Ficando mais ou menos assim:

<div class="row">

    <div class="col-md-3">

        <div class="card" style="width: 18rem;">

            <img src="..." class="card-img-top" alt="...">

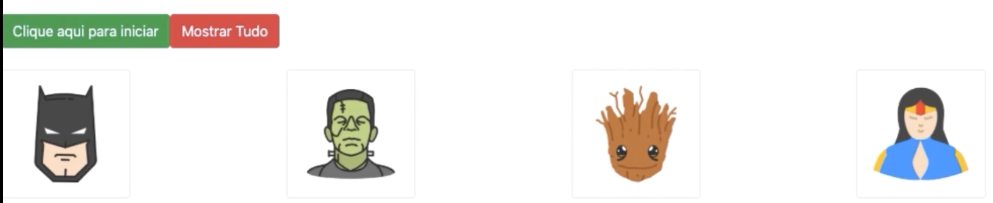
        </div>

    </div>

Inclua os seus 4 ‘heróis’

Resultado do professor:

Exclua este código

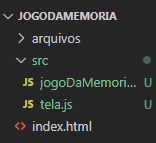


Meu resultado:



IMAGENS DINÂNICAS

A estrutura deve estar desta forma agora:



Dentro de src crie dois arquivos:

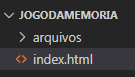
touch Tela.js

touch JogoDaMemoria.js

Agora crie um novo diretório:

mkdir src

Sua estrutura esta desta forma:



Será responsável por toda a lógica do jogo

Será responsável pela tela do jogo

class JogoDaMemoria{

}

class Tela{

}

Conteúdo inicial do arquivo JogoDaMemoria.js e do Tela.js

Dentro da raiz do projeto – JogoDaMemoria – junto com o index.html **crie** o arquivo index.js

este arquivo será o responsável por chamar os outros .js e agrupar tudo

Teste o jogo, através, do index.html, deve estar tudo funcionando normalmente

- Colocar o defer no 1º script foi essencial para o funcionamento do projeto

No arquivo ./index.html, entre os heads, digite:

<script defer src="./index.js"></script>

<script src="./src/JogoDaMemoria.js"></script>

<script src="./src/Tela.js"></script>

No ./index.html ainda:

 <div class="row">

        <div class="col-md-3">

            <div class="card" style="width: 18rem;">

                <img src="./arquivos/america.svg" class="card-img-top" alt="...">

            </div>

        </div>

...

Selecione o script de qualquer um do heróis e RECORTE, neste caso recortarei o 1º

No arquivo ./src/Tela.js, dentro da classe principal,

1 – Crie uma função obterCodigoHtml(item){ }

2 – Esta função deve retornar o código html recortado anteriormente

A class row deve ficar ‘vazia’

<div class="row">

</div>

Esta função deve ter um argumento chamado ‘item’

class Tela{

    static obterCodigoHtml(item){

        return `

        <div class="col-md-3">

            <div class="card" style="width: 18rem;">

                <img src="./arquivos/america.svg" class="card-img-top" alt="...">

            </div>

        </div>

        `

    }

}

1

2

Observe as crases

Faça as seguintes alterações e inclusão neste código:

    <img src="${item.img}" name="${item.nome}" class="card-img-top" alt="...">

</div>

<br />

Estes item.img e item.name ainda não foram criados

Logo abaixo do:

static obterCodigoHtml(item){

...

}

Criaremos uma nova função:

static alterarConteudoHtml(codigoHtml){

    const conteudo = document.getElementById(ID\_CONTEUDO);

    conteudo.innerHTML = codigoHtml;

};

Este ID\_CONTEUDO está definido antes da class Tela:

const ID\_CONTEUDO = 'conteudo';

class Tela{

...

Esta palavra ‘conteudo’ deverá ser um ID no documento index.html



Agora, no arquivo, ./src/index.js

// Criando um objeto heroi

const heroi = {

    // sempre relativo ao index.html

    img: './arquivos/america.svg',

    nome: 'america'

};

// Usando as funções definidas no arquivo ./src/Tela.js

// as funções são estáticas – static – por isso é só usar o

// nome da classe ( Tela ) no arquivo Tela.js

const codigoHtml = Tela.obterCodigoHtml(heroi);

Tela.alterarConteudoHtml(codigoHtml);

Testando o index.html, temos:



Se alterarmos:

Tela.alterarConteudoHtml(codigoHtml);

Para:

Tela.alterarConteudoHtml(codigoHtml.concat(codigoHtml).concat(codigoHtml));

Teremos, 3 capitães américas:



Usamos a função concat do objeto String codigoHtml e concatenamos 3 vezes o código retornado ( return ). As posições de cada imagem ficou a cargo do bootstrap

Isso mostra que nosso código está funcionando. Mas vamos continuar evoluindo...

No arquivo ./src/Tela.js

...

static gerarStringHtmlPelaImagem(itens){

    // Para cada item de uma lista, vai executar a função

    // obterCodigoHtml e ao final, vai concatenar tudo em

    // uma string

    // A função de array map, retornará a lista concatenada

    return itens.map(Tela.obterCodigoHtml).join('');

}

static atualizarImagens(itens){

    const codigoHtml = Tela.gerarStringHtmlPelaImagem(itens);

    Tela.alterarConteudoHtml(codigoHtml);

}

...

No arquivo ./index.js, apague as seguintes linhas:

...

const codigoHtml = Tela.obterCodigoHtml(heroi);

Tela.alterarConteudoHtml(codigoHtml.concat(codigoHtml).concat(codigoHtml));

Código ‘final’ de ./src/Tela

const ID\_CONTEUDO = 'conteudo';

class Tela{

    static obterCodigoHtml(item){

        return `

        <div class="col-md-3">

            <div class="card" style="width: 18rem;">

            <img src="${item.img}" name="${item.nome}" class="card-img-top" alt="...">

        </div>

        <br />

        </div>

        `

    };

    static alterarConteudoHtml(codigoHtml){

        const conteudo = document.getElementById(ID\_CONTEUDO);

        conteudo.innerHTML = codigoHtml;

    };

    static gerarStringHtmlPelaImagem(itens){

        return itens.map(Tela.obterCodigoHtml).join('');

    }

    static atualizarImagens(itens){

        const codigoHtml = Tela.gerarStringHtmlPelaImagem(itens);

        Tela.alterarConteudoHtml(codigoHtml);

    }

};

E digite:

Tela.atualizarImagens([

    heroi,

    heroi,

    heroi,

    heroi

]);

Código final de ./index.js

const heroi = {

    img: './arquivos/america.svg',

    nome: 'america'

};

Tela.atualizarImagens([

    heroi,

    heroi,

    heroi,

    heroi

]);

Resultado:



O arquivo ./index.js será totalmente reformulado, porém vamos trabalhar agora com o arquivo JogoDaMemoria.js

class JogoDaMemoria{

    // O constructor está recebendo um objeto

    constructor({tela}){

        this.tela = tela;

        this.heroisIniciais = [

            {img: './arquivos/america.svg', name: 'Capitão América'},

            {img: './arquivos/batman.svg', name: 'Batman'},

            {img: './arquivos/flash.svg', name: 'Flash'},

            {img: './arquivos/wolverine.svg', name: 'Wolverine'}

        ];

    };

    inicializar(){

        this.tela.atualizarImagens(this.heroisIniciais);

    };

};

E o ./index.js é totalmente sobrescrito com:

const dependencias = {

    tela: Tela

};

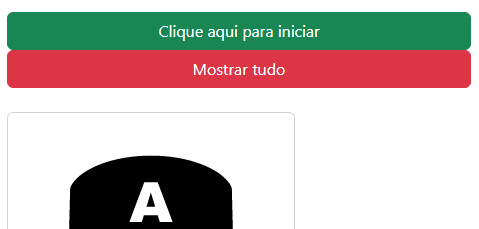
const jogoDaMemoria = new JogoDaMemoria(dependencias);

jogoDaMemoria.inicializar();

Resultado:



EMBARALHAR



O embaralhamento dos cards se dará quando clicado o botão jogar. Onde

no ./index.html tem o ID ‘jogar’

<button type="button" class="btn btn-success" id="jogar">Clique aqui para iniciar</button>

Vamos iniciar alterando o arquivo ./src/Tela.js, ele é reponsável por passar todos os dados visuais pro ./index.html, Tela, não pode ter nenhuma lógica, mas/é aqui que o professor coloca a linkagem dos Ids e Class passando a função a ser executada aqui pela ./src/JogoDaMemoria.js

No arquivo ./src/Tela.js

const ID\_BTN\_JOGAR = 'jogar';

1º no topo do arquivo definimos uma constante que guarda o nome do ID que representa o botão

2º crio uma função específica para atrelar a constante ID\_BTN\_JOGAR a uma outra constante btnJogar e em seguida registrar a função onClick nele e atribuir a função a ser executado quando clicado

static configurarBotaoJogar(funcaoOnClick){

    const btnJogar = document.getElementById(ID\_BTN\_JOGAR);

    btnJogar.onclick = funcaoOnClick;

}

Interessante essa abordagem do professor, como variáveis e constantes dentro de uma função tem escopo local, a memória usada é limpa na finalização de cada click e recriada quando o botão for clicado.

Essa função, que não precisa ter o mesmo nome, é claro.

No arquivo ./src/JogoDaMemoria.js

Vamos criar, de fato, a função que será usada ao clicar no botão para iniciar o jogo

Só para testar, criamos uma função qualquer

...

    inicializar(){

        this.tela.atualizarImagens(this.heroisIniciais);

        this.tela.configurarBotaoJogar( this.jogar.bind(this) );

    };

    jogar(){

        console.log('clicou');

    };

...

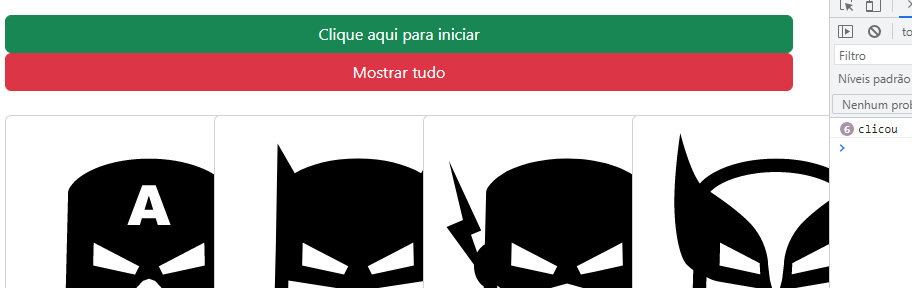
chamamos a função configurarBotaoJogar pela variável this.tela e passamos como argumento a função jogar recém criada, passamos a função e não executamos a função. Chamamos uma função base das function, bind, para definir qual o contexto do this, que neste caso refere-se a JogoDaMemoria.js e não Tela,js

Em inicializar...

Entãsis podemos imaginar que Tela.js de onde partiu o script terá, só na chamada da função o this de JogoDaMemoria.js

A cada click

Resultado:



Continuando em ./src/JogoDaMemoria.js

Vamos criar o método embaralhar, que será o coração do projeto, manipulará, mostrará e ocultará as cartas:

...

};

embaralhar(){

    ...

}

jogar(){

    console.log('clicou');

};

...

1º - Criar um array de constantes com nosso heroisInicais – Na Programação funcional não alteramos a ‘variável’ em si e sim em uma cópia

...

embaralhar(){

    const copias = this.heroisIniciais

}

...

Como sabemos que copias é um array? Porque ele recebe this.heroisIniciais que é um array, neste caso array de objetos herois

this.heroisIniciais = [

    {img: './arquivos/america.svg', name: 'Capitão América'},

    {img: './arquivos/batman.svg', name: 'Batman'},

    {img: './arquivos/flash.svg', name: 'Flash'},

    {img: './arquivos/wolverine.svg', name: 'Wolverine'}

];

2º - Duplicamos a quantidade de heróis fazendo uma cópia do array this.heroisIniciais dentro do array copias:

...

embaralhar(){

    const copias = this.heroisIniciais.concat(this.heroisIniciais);

}

...

Dentro de copias agora tem o dobro de heróis:

...

copias = [

    {img: './arquivos/america.svg', name: 'Capitão América'},

    {img: './arquivos/batman.svg', name: 'Batman'},

    {img: './arquivos/flash.svg', name: 'Flash'},

    {img: './arquivos/wolverine.svg', name: 'Wolverine'},

    {img: './arquivos/america.svg', name: 'Capitão América'},

    {img: './arquivos/batman.svg', name: 'Batman'},

    {img: './arquivos/flash.svg', name: 'Flash'},

    {img: './arquivos/wolverine.svg', name: 'Wolverine'}

];

...

3º - Para organizarmos o código melhor vamos colocar a função concat na linha abaixo:

...

embaralhar(){

    const copias = this.heroisIniciais.concat(this.heroisIniciais);

}

...

Tem o mesmo resultado que:

...

embaralhar(){

    const copias = this.heroisIniciais

    .concat(this.heroisIniciais);

}

...

Como o professor vai fazer uma sequência de comandos bem grande é melhor ficar um embaixo do outro

4º - Vamos usar um método do arquivo ./src/Tela.js para atualizar a visão dos heróis no navegador:

...

embaralhar(){

    const copias = this.heroisIniciais

    .concat(this.heroisIniciais)

    this.tela.atualizarImagens(copias);

}

...

Logo abaixo no método jogar() chamamos o método embaralhar:

...

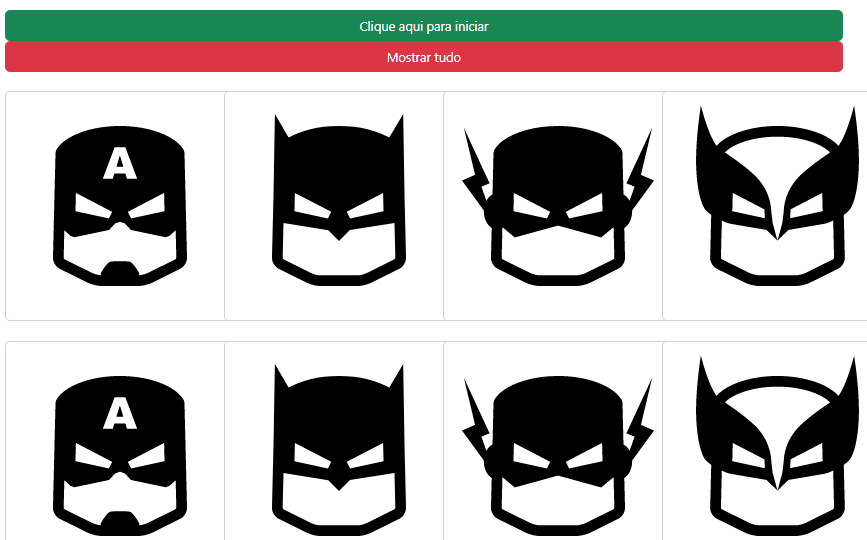
jogar(){

    this.embaralhar();

};

...

Clicando no botão de ID jogar e Value ‘Clique aqui para iniciar’, uma copia dos heróis aparecem:



5° Vamos embaralhar as imagens de heróis:

...

embaralhar(){

    const copias = this.heroisIniciais

    .concat(this.heroisIniciais)

    .sort(() => Math.random() - 0.5)

    this.tela.atualizarImagens(copias);

}

...

.sort() pega todos os itens do array, transforma em strings e em seguida classifica cada um usando uma arrow function que retorna um número randômico que tem como resultado embaralhar os elementos

O professor não fez deste jeito. Ele criou um novo elemento chamado id dentro de cada objeto do array heroisIniciais

    {

        img: './arquivos/america.svg',

        name: 'Capitão América',

id: 0.00340430

    }

    {

        img: './arquivos/america.svg',

        name: 'Capitão América'

    }

this.heroisIniciais = [

    {img: './arquivos/america.svg', name: 'Capitão América'},

    {img: './arquivos/batman.svg', name: 'Batman'},

    {img: './arquivos/flash.svg', name: 'Flash'},

    {img: './arquivos/wolverine.svg', name: 'Wolverine'}

];

Mas ele fez isso dinamicamente usando a função .map(...)

Observe o código abaixo:

1

embaralhar(){

    const copias = this.heroisIniciais

    .concat(this.heroisIniciais)

    .map(item => {

        return Object.assign({}, item, { id: Math.random() / 0.5})

    })

    .sort(() => Math.random() - 0.5)

    this.tela.atualizarImagens(copias);

}

2

4

3

7

5

6

1 – Estamos trabalhando em cima de um array de objetos heroisIniciais

2 – Neste momento temos 8 objetos no array copias

3 – .map retorna um novo array a partir das alterações feitas pela arrow function do item 4

4 – O argumento da função .map é uma arrow function que inicializa um object vazio que é preenchido com cada um dos elementos item (cada heróis) incluídos com o id e o retorna (return)

5 – item representa cada um do objetos heroisIniciais dentro do array copias

6 – Em cada um dos objetos do array copias o código vai incluir um novo elemento chamado id com um valor aleatório ( randômico )

7 – Object vazio que receberá sobreporá o objetos de heroisInciais com a devida inclusão do id:

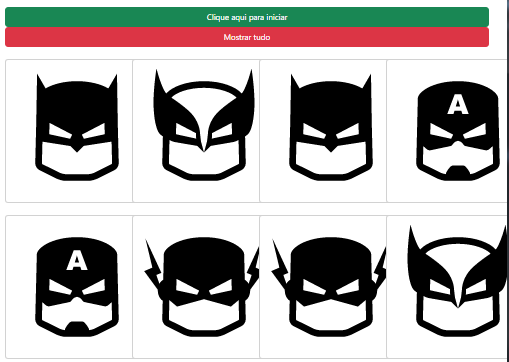
{

   img: './arquivos/america.svg',

   name: 'Capitão América',

id: 0.00340430

}



Por fim a linha:

.sort(() => Math.random() - 0.5)

embaralha a ‘posição’ de cada objeto no array copias e em seguida usa a função:

this.tela.atualizarImagens(copias);

Para atualizar a visualização dos heróis na navegador