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>

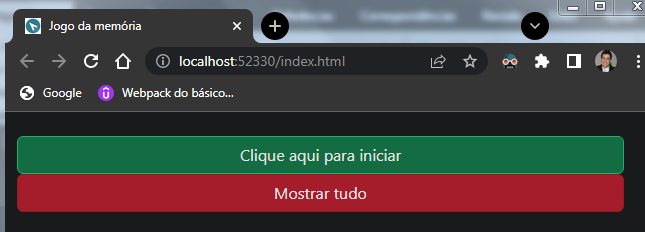
<br/>

<div class="row container-fluid">

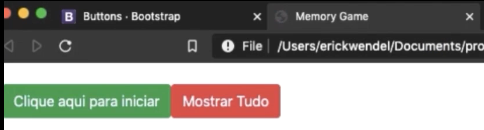
    <button type="button" class="btn btn-success" id="jogar">Clique aqui para iniciar</button>

    <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>

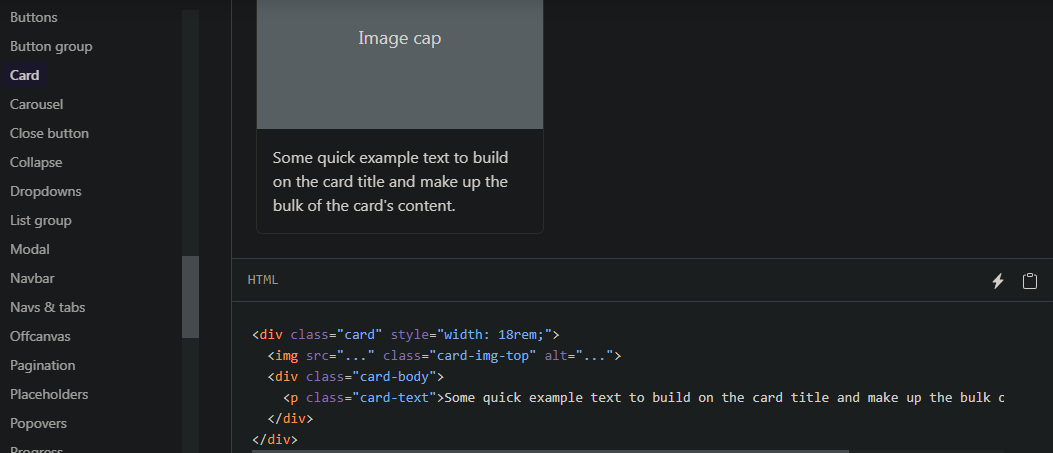
    </div>

</div>

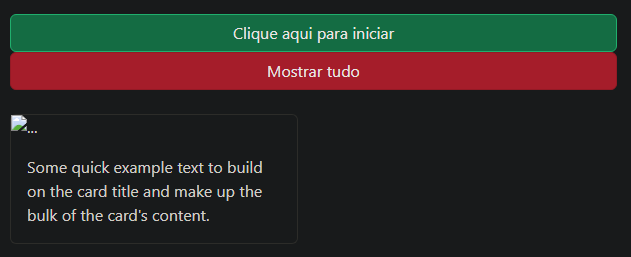
...

Copie o código html e cole em seu código

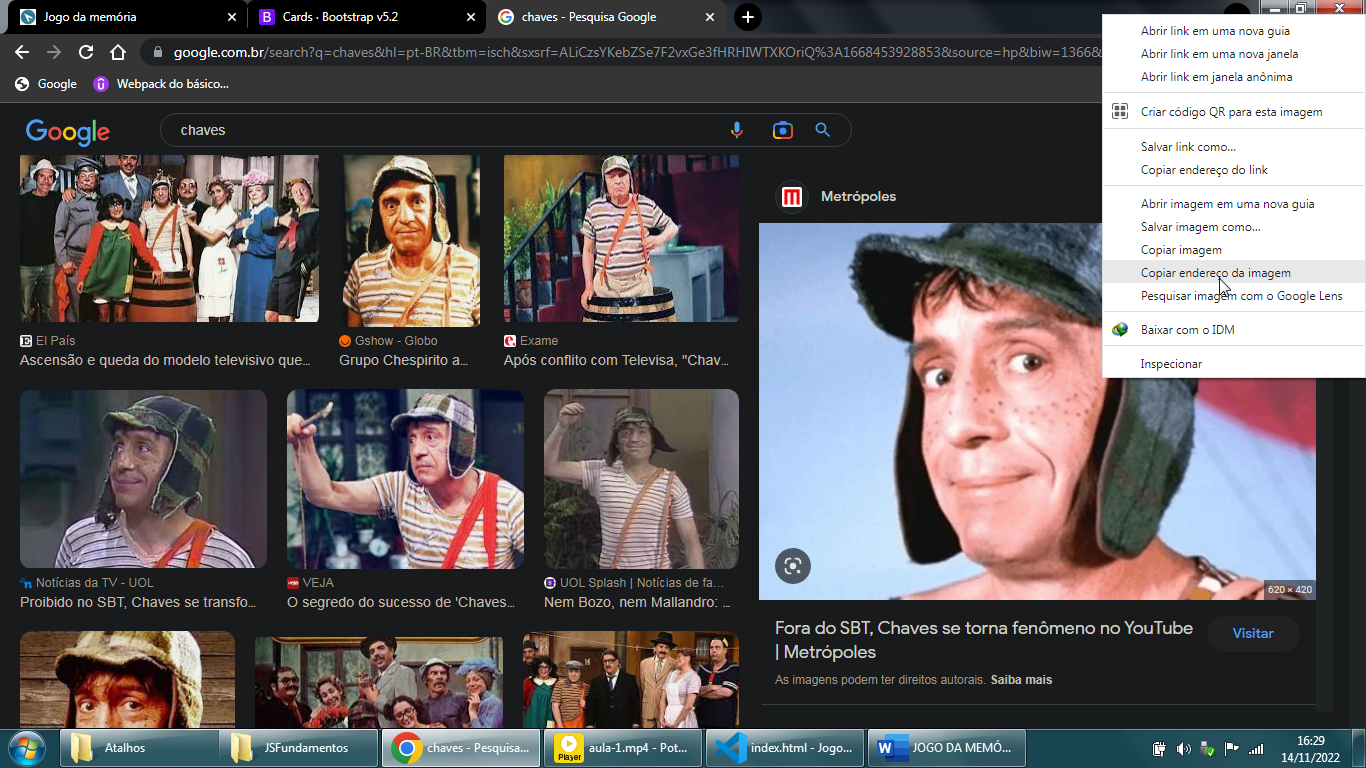
Na documentação do bootstrap -> card – Selecione algo do tipo:

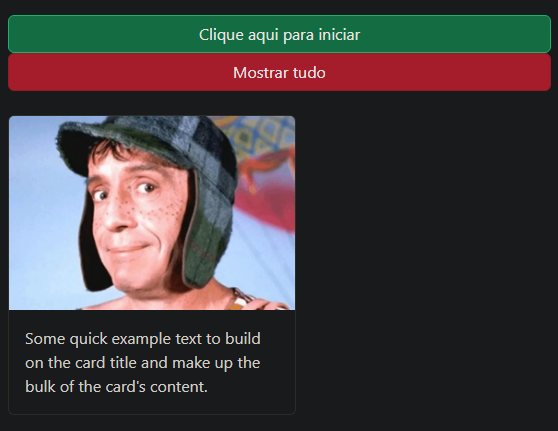


Resultado:



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="...">

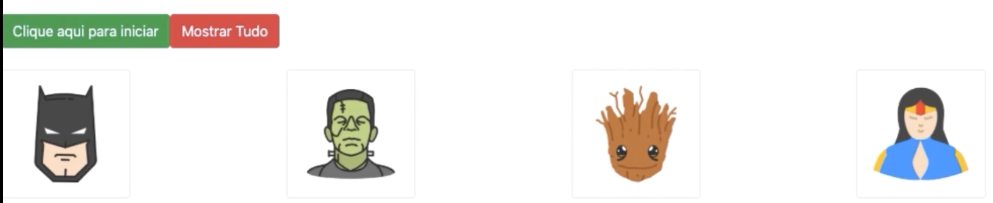
        </div>

    </div>

Inclua os seus 4 ‘heróis’

Resultado do professor:

Exclua este código

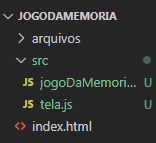


Meu resultado:



IMAGENS DINÂNICAS

A estrutura deve estar desta forma agora:



Dentro de src crie dois arquivos:

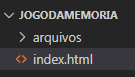
touch Tela.js

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

};

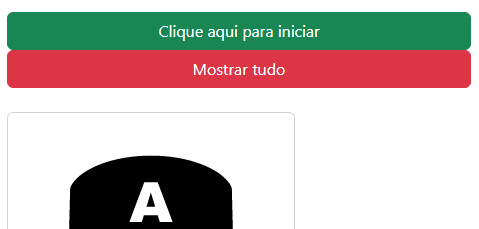
const jogoDaMemoria = new JogoDaMemoria(dependencias);

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');

    };

...

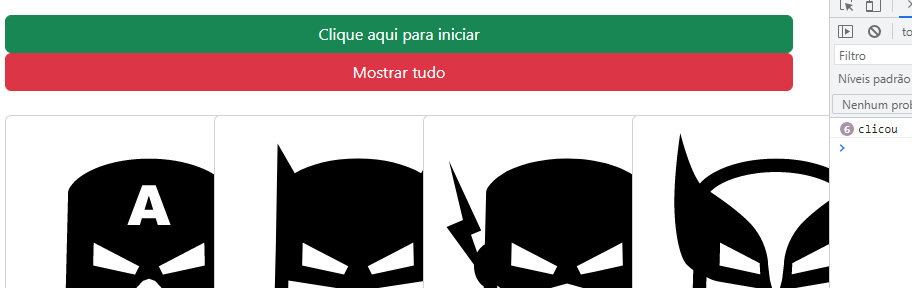
chamamos a função configurarBotaoJogar pela variável this.tela e passamos como argumento a função jogar recém criada, passamos a função e não executamos a função. Chamamos uma função base das function, bind, para definir qual o contexto do this, que neste caso refere-se a JogoDaMemoria.js e não Tela,js

Em inicializar...

Entãosis 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:

...

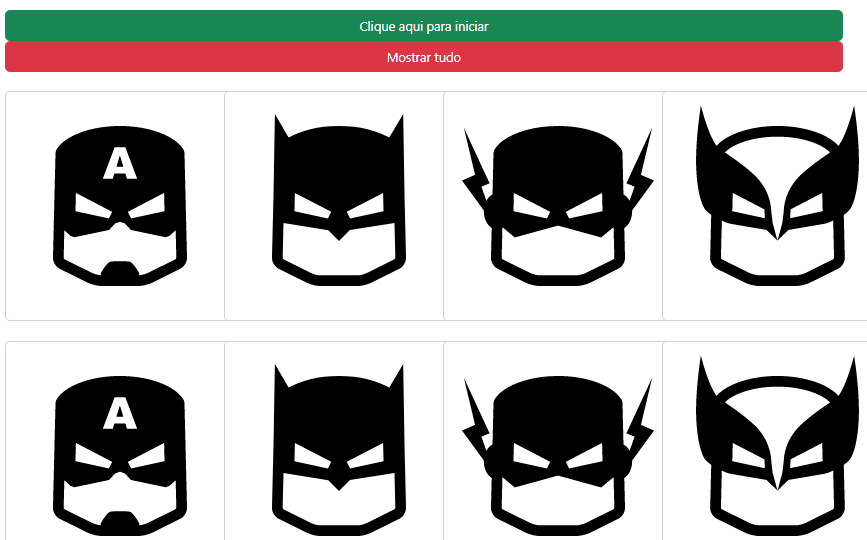
jogar(){

    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:

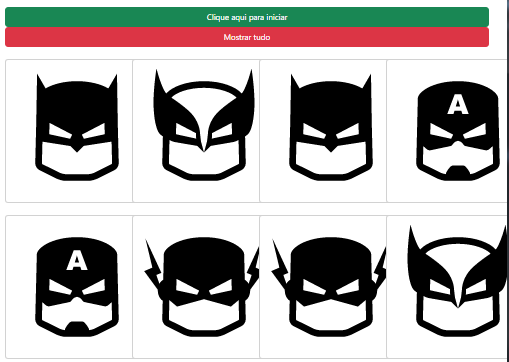
{

   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

Vale ressaltar que Math.random() não é uma forma adequada de se criar ids. Com ele, sempre vai existir a possibilidade de gerar valores iguais

Esconder imagens e definir imagem padrão das imagens escondidas

1 – No arquivos JogoDaMemoria.js, no construtor da classe vamos incluir:

...

    this.iconePadrao = './arquivos/ninja.svg'

    this.heroisEscondidos = []

};

...

O endereço da imagem padrão das imagens escondidas

Um array para conter os heroisIniciais configurados com a imagem padrão das imagens escondidas

**herois** – o array heroisIniciais, ou uma cópia deles

**const heroisOcultos** – Programação funcional... uma cópia de heroisIniciais, porém, uma cópia alterada pela função .map

**{nome, id}** uma desestruturação de heroisIniciais, pegando apenas nome e id, e não o item img

2 – Criar função esconderHerois:

...

esconderHerois(herois){

    const heroisOcultos = herois.map(({nome, id}) => ({

        id,

        nome,

        img: this.iconePadrao

    }))

    this.tela.atualizarImagens(heroisOcultos)

    this.heroisEscondidos = heroisOcultos

}

...

Por omitir a atribuição destas propriedades, o js usa do conceito de Object Short Syntax, onde se a propriedade possuir o nome idêntico a um argumento do método, esta propriedades recebe o mesmo valor que o argumento do método

**id,**

**nome,**

**img: this.iconePadrao** – Com a cópia, por causa da desestruturação realizada com {nome, id}, não traz a propriedade img, ela ( img ) é criada explicitamente recebendo o enderço da imagem padrão usada quando os heróis estão escondidos

**this.tela.atualizarImagens(heroisOcultos)** – exibe os heróis, mas com a imagem this.iconePadrao, assim os heróis ficam escondidos

**this.heroisEscondidos = heroisOcultos** – Armazena esta cópia alterada de heroisIniciais em variável criada no construtor que poderá ser usada dispois

**Minha observação:** sempre que for esconder as imagens, todo este script será executado, não seria melhor parte dele já estar pronto, tipo a cópia dos heroisInciais, que alias, nem precisaríamos trabalhar com um cópia e simplesmente atribuir a imagem iconePadrao diretamente

3 – Dentro do método embaralhar:

...

    this.tela.atualizarImagens(copias)

    setTimeout(() => {

        this.esconderHerois(copias)

    }, 1000);

}

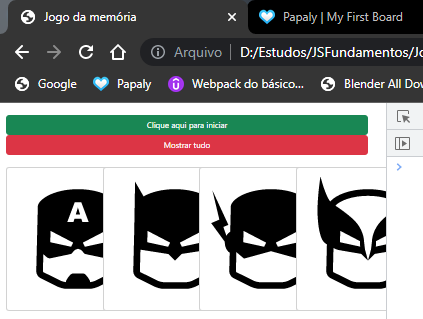
...

usar a função **setTimeout** para definir 1 segundo de espera até chamar a função **this.esconderHerois(copias).** Aqui, **copias** é uma copia de heroisIniciais

Encontrar combinação

Antes, vamos “corrigir” um pequeniníssimo “probleminha”

Em algum momento, principalmente quando estiver debugando uma página da net, os heróis ficam ‘empilhados’



O código reponsável por isso está no arquivo ./src/Tela.js

Aqui para ser mais exato

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>

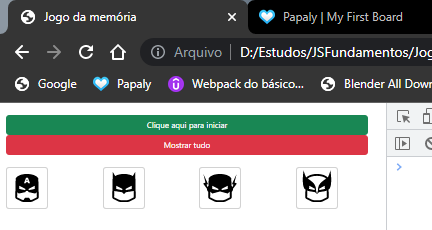
        <br />

        </div>

        `

    };

Altere o 18rem para 50% e terá este resultado:



ai vc decide qual resultado vc prefere nestas condições, lembre-se, o seu trabalho poderá ser acessado em um celular ou tablet

Encontrar combinação

Primeiro vamos fazer um teste. Eu quero que cada vez que for clicado em um “figura” uma alerta surge mostrando o nome e o id da mesma. Continuando em ./src/Tela

Inclua este código

static obterCodigoHtml(item){

    return `

    <div class="col-md-3">

        <div class="card" style="width: 18rem;" onclick="window.alert('id: ${item.id}, nome: ${item.nome}')" >

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

    </div>

    <br />

    </div>

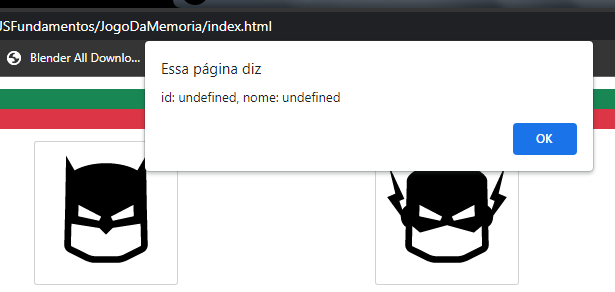
    `

};

Resultado:

id: undefined era esperado já que o id de cada heróis é definido somente quando clicado o botão ‘click aqui para iniciar’ de id ‘jogar’

nome: undefined deveria exibir o nome do herói. Vamos ver o código do click dos heróis que foi definido anteriormente, acima



1 - Aqui está chamando o item.nome

static obterCodigoHtml(item){

    return `

    <div class="col-md-3">

        <div class="card" style="width: 18rem;" onclick="window.alert('id: ${item.id}, nome: ${item.nome}')" >

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

    </div>

    <br />

    </div>

    `

};

2 – item é um parâmetro da função chamada

3 – Para resumir, item é o nome de cada herói que receberá o click e é definido em ./src/JogoDaMemoria.js

...

    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'}

        ];

        this.iconePadrao = './arquivos/ninja.svg'

        this.heroisEscondidos = []

    };

...

4 – Aqui o erro, a propriedade é chamada de ‘name’ e não ‘nome’

5 – Como estamos usando o português no código, em vez de mudar o ‘nome’ por ‘name’, vamos mudar o ‘name’ para ‘nome’

...

this.heroisIniciais = [

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

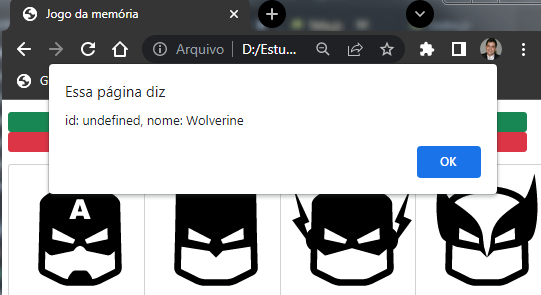
    {img: './arquivos/batman.svg', nome: 'Batman'},

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

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

];

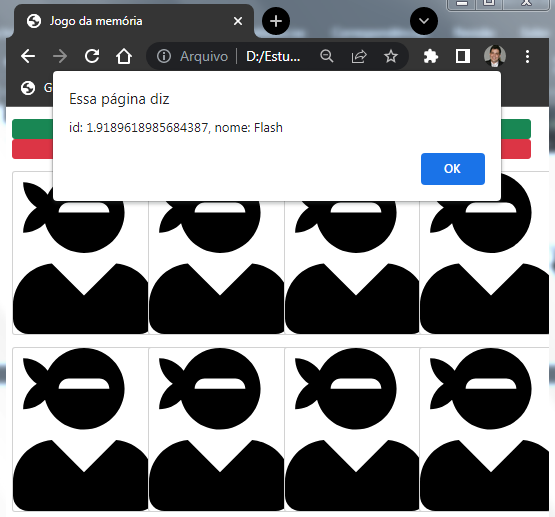
...



Resultado:

Antes de clicar em ‘click aqui para iniciar’, clicando em cada herói será visualizado o seu nome:

O ‘id’ ainda não aparece



E depois de clicado em ‘click aqui para iniciar’

- É criado uma cópia de cada herói

- É criado a propriedade ‘id’ para cada um dos heróis

- A posição de cada herói é ‘embaralhada’

- Depois de 1 segundo os heróis são ocultos

Mais um probleminha aqui, mesmo os heróis estando ocultos, a cada click em cada imagem o ‘id’ e o ‘nome’ de cada herói ainda aparece

Esta função ‘onClick’ chamada para definir este ‘alert’ está no arquivo ./src/Tela.js, intãosis precisamos leva-la para ./src/JogoDaMemoria.js onde está definido a lógica do jogo e as funções que definem as ações

- Aqui é um texto concatenado com variáveis

- Aqui é passado 2 argumentos

Em ./src/Tela.js ‘definiremos’ esta função

static configurarBotaoVerificarSelecao(funcaoOnClick){

    window.verificarSelecao = funcaoOnClick;

}

Neste mesmo arquivo trocar:

...

<div class="card" style="width: 18rem;" onclick="window.alert('id: ${item.id}, nome: ${item.nome}')" >

...

por:

…

<div class="card" style="width: 18rem;" onclick="window.verificarSelecao('${item.id}', '${item.nome}')" >

…

...

this.tela.configurarBotaoVerificarSelecao(this.verificarSelecao.bind(this));

...

...

verificarSelecao(id, nome){

    const item = { id, nome};

    alert(`id: ${item.id}, nome: ${item.nome}`);

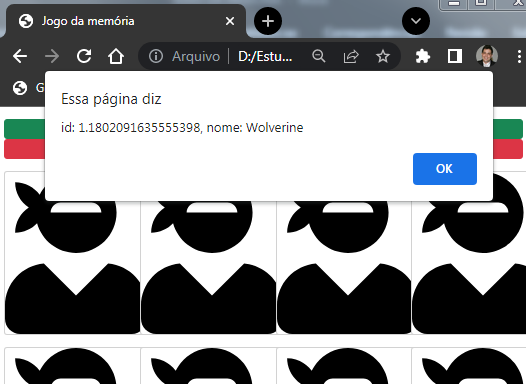
}

...

Em ./src/JogoDaMemoria.js criar a função:

Na função inicializar() incluímos uma chamada a função inicial ‘configurarBotaoVerificarSelecao’ que estará em ./src/Tela.js mas a ação em sí será a ‘verificarSelecão’

Resultado:



O probleminha aqui continua, mesmo os heróis estando ocultos, a cada click em cada imagem o ‘id’ e o ‘nome’ de cada herói ainda aparece, mas na verdade este alerta foi criado apenas para ver se a função verificarSelecao lá no arquivo ./src/JogoDaVelha.js está sendo chamado corretamente. É só tirar o alert e trocar pela lógica que deve ter.

No construtor da classe JogoDaVelha, no arquivo ./src/JogoDaVelha.js incluímos:

...

    this.heroisSelecionados = []

...

Ainda em ./src/JogoDaVelha.js

...

verificarSelecao( id, nome ){

    const item = { id, nome};

    const heroisSelecionados = this.heroisSelecionados.length

    switch(heroisSelecionados){

        case 0:

            this.heroisSelecionados.push(item)

        break

        case 1:

            const [opcao1] = this.heroisSelecionados

            this.heroisSelecionados = []

            if( opcao1.nome === item.nome &&

                opcao1.id !== item.id

            ){

                alert(`Combinação correta: ${item.nome}`)

                return

            }

            alert(`Combinação incorreta: ${item.nome} !== ${opcao1.nome} `)

        break

    }

}

...

Objeto clicado

Técnica funcional, não agir sobre um variável, fazer uma cópia

Verificar quantos heróis já foram selecionados, primeira vez tem que ser zero

Se for zero, inclui um herói

Se não for zero

No JS vc pode criar uma variável desta forma

O herói que foi incluído aqui, um objeto

Como memória é um jogo de pares, podemos limpar a memória zerando os heroisSelecionados

Comparando o herói clicado anteriormente com o atual herói clicado. Comparar o id impede que o usuário click duas vezes no mesmo card e seja válido

Se a combinação for correta...

...faça com que o script pare

A combinação não formando par este alerta é ativado mesmo sem um else, já que se houvesse par o return é usado e esta linha não seria executada

Abaixo o código sem comentários:

verificarSelecao(id, nome){

    const item = { id, nome};

    const heroisSelecionados = this.heroisSelecionados.length

    switch(heroisSelecionados){

case 0:

            this.heroisSelecionados.push(item)

        break

        case 1:

            const [opcao1] = this.heroisSelecionados

            this.heroisSelecionados = []

            if(opcao1.nome === item.nome &&

                opcao1.id !== item.id

            ){

                alert(`Combinação correta: ${item.nome}`)

                return

            }

            alert(`Combinação incorreta: ${item.nome} !== ${opcao1.nome} `)

        break

    }

}

Exibir herói

No arquivo ./src/JogoDaMemoria.js

No método verificarSelecao(id, nome) {...} em dois momentos será enviado um objeto com os heroisSelecionados, que será o argumento deste parâmetro

O objeto no parâmentro nomeDoHeroi será um objeto heroiSelecionados do método verificarSelecao

...

exibirHerois(nomeDoHeroi){

    const {img} = this.heroisIniciais.find(

        ({nome})=>nomeDoHeroi === nome

    )

}

...

img receberá a imagem do herói desestruturado de heroisIniciais

Vamos usar o método find para procurar qual nome de herói no this.heroisIniciais que corresponde ao nome do herói vindo do parâmetro nomeDoHeroi

Uma arrow function de apenas um parâmetro, nome, resultado de uma desestruturação, e como só vai haver uma linha de lógica, o resultado desta linha única **return** ( literalmente ) o resultado que é desestruturado retornando o **img** do herói.

O parâmetro, nomeDoHeroi não pode ser utilizado para retornar o **img** porque ele foi preenchido com uma versão sem o **img**

... this.tela.exibir(nomeDoHeroi, img)

E para encerrar a função...

...

exibirHerois(nomeDoHeroi){

    const {img} = this.heroisIniciais.find(

        ({nome})=>nomeDoHeroi === nome

    )

    this.tela.exibirHerois(nomeDoHeroi, img)

}

...

Vale lembrar que, nomeDoHeroi, conterá apenas UM nome, mas esse nome estará presente em DUAS imagens, definindo assim o PAR. Por isso o professor inclui o ID nos objetos Herois para definir um diferencial e evitar que o usuário clique na mesma imagem e o jogo retorne que ele formou a PAR correto

Mais um detalhe, está função, this.tela.exibirHerois(nomeDoHeroi, img) ainda não foi definida no arquivo ./src/Tela.js. Vamos fazer isso agora....

No arquivo ./src/Tela.js

...

    static exibirHerois( nomeDoHeroi, img){

        const elementosHtml = document.getElementsByName(nomeDoHeroi)

        elementosHtml.forEach( item => (item.src = img))

    }

...

1

3

2

5

4

1 – nomeDoHeroi conterá apenas UM nome de herói

2 – elementosHtml receberá um...

3 – ...CONJUNTO de todos os elementos do index.html que possui aquele nome

4 – O JS passará por cada elemento e...

5 – ...incluirá o a imagem correspondente aos dois heróis encontrados

Este método ( exibirHerois ) deverá ser chamado no método verificarSelecao ( no arquivo JogoDaMemoria.js ) QUE EXIBI AS IMAGENS CLICADAS APENAS QUANDO AS DUAS TEM O MESMO NOME E ID

- Agora falta alterar o arquivo ./JogoDaVelha.js, método verificarSelecao para realizar a exibição das imagens do herói caso estas tenham o mesmo nome e id

Método alterado:

...

verificarSelecao(id, nome){

    const item = { id, nome};

    const heroisSelecionados = this.heroisSelecionados.length

    switch(heroisSelecionados){

        case 0:

            this.heroisSelecionados.push(item)

        break

        case 1:

            const [opcao1] = this.heroisSelecionados

            this.heroisSelecionados = []

            if(opcao1.nome === item.nome &&

                opcao1.id !== item.id

            ) {

                this.exibirHerois(item.nome);

                return

            }

            alert(`Combinação incorreta: ${item.nome} !== ${opcao1.nome} `)

        break

    }

}

...

Método anterior:

...

verificarSelecao(id, nome){

    const item = { id, nome};

    const heroisSelecionados = this.heroisSelecionados.length

    switch(heroisSelecionados){

case 0:

            this.heroisSelecionados.push(item)

        break

        case 1:

            const [opcao1] = this.heroisSelecionados

            this.heroisSelecionados = []

            if(opcao1.nome === item.nome &&

                opcao1.id !== item.id

            ){

                alert(`Combinação correta: ${item.nome}`)

                return

            }

            alert(`Combinação incorreta: ${item.nome} !== ${opcao1.nome} `)

        break

    }

}

...

Este ALERT simplesmente será trocado pela chamada do método EXIBIRHEROIS

Exibir Mensagem de Erro ou Acerto

- No arquivo ./index.html, depois da última tag </div>, incluir:

Procure Alert na documentação do BootStrap

Classes, class, do BootSrap

Inicialmente está <div> não estará visível

...

<div class= "row container-fluid" >

    <div id="mensagem" class= "alert alert-success invisible" role="alert">

        Combinação!!!

    </div>

</div>

...

Incluindo um ID

Esta class, alert-success será exibida quando o par de heróis combinar e será alterado para alert-danger quando não combinar

Este texto não será exibido

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

...

const ID\_MENSAGEM = 'mensagem';

const CLASSE\_INVISIBLE = 'invisible';

...

Constantes que definirão o nome da tag a ser referida, no caso um id

1

...

const MENSAGENS = {

    sucesso: {

        texto: 'Combinação correta',

        classe: 'alert-success'

    },

    erro: {

        texto: 'Combinação INCORRETA',

        classe: 'alert-danger'

    }

}

...

Esta constante ( 1 ) definirá a mensagem a ser exibida quando o usuário acerta um par de heróis ( 2 ) ou não ( 3 ) e a sua aparência em cada um dos casos ( 4 )

2

4

Ainda em ./src/Tela.js

1 – Define o padrão do parâmetro para true

2 – Define o elemento html a ser usado

3 – Remove a classe alert-success ou alert-danger

4 – Adiciona uma classe, class

5 – Define um texto para a mensagem

6 – Remove a classe, class, invisible – É chamada várias vezes, mas uma só já serveria

3

4

1

...

static exibirMensagem(sucesso = true){

    const elemento = document.getElementById(ID\_MENSAGEM)

    if(sucesso){

        elemento.classList.remove(MENSAGENS.erro.classe)

        elemento.classList.add(MENSAGENS.sucesso.classe)

        elemento.innerText = MENSAGENS.sucesso.texto

    }else{

        elemento.classList.remove(MENSAGENS.sucesso.classe)

        elemento.classList.add(MENSAGENS.erro.classe)

        elemento.innerText = MENSAGENS.erro.texto

    }

    elemento.classList.remove(CLASSE\_INVISIBLE)

}

...

2

3

4

5

3

4

5

6

Refatorando o setTimeout()

Vamos criar o arquivo ./src/Util.js

Conteúdo:

class Util{

    static timeOut(tempo){

        return new Promise(resolve => setTimeout(resolve, tempo))

    }

}

4

5

3

2

1

1 – Esta função retorna uma Promise

2 – Vamos trabalhar apenas com a função, resolve, chamada quando a Promise retorna algo com sucesso. resolve aqui, é o único parâmetro da arrow function

3 – a arrow function return setTimeout(

4 – resolve é a função que será chamada a cada

5 – tempo, em quanto tempo em milissegundos a função resolve será chamada

No arquivo ./index.html , incluir a chamada da classe Util.js:

…

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

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

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

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

…

É importante que a classe Util.js seja chamada antes de JogoDaMemoria.js e Tela.js porque estas duas classe poderão usar a classe Util,js

No arquivo ./index.js inclua Util.js como uma das dependências do jogo:

...

const dependencias = {

    tela: Tela,

    util: Util

};

const jogoDaMemoria = new JogoDaMemoria(dependencias);

...

Perceba que foi necessário apenas incluir ‘Util’ na

constante dependências, que automaticamente, util, também foi enviada ao JogoDaMemoria

O arquivo ./src/JogoDaMemoria.js deve receber etsa classe Util.js

...

constructor({tela}){

    this.tela = tela;

    this.heroisIniciais = [

...

...

constructor({tela, util}){

    this.tela = tela;

    this.util = util;

    this.heroisIniciais = [

...

Antes

Depois

Continuando em ./src/JogoDaMemoria.js, no método embaralhar, trocamos:

...

setTimeout(() => {

    this.esconderHerois(copias)

}, 1000);

...

Por:

...

await this.util.timeOut(1000)

this.esconderHerois(copias)

…

Para await funcionar, na função onde ela é usada, no caso a função embaralhar, devemos adicionar a palavra chave async

...

async embaralhar(){

    const copias = this.heroisIniciais

...

await ( declarada em uma função ) diz que o código só seguira quando houver um retorno da função seguinte, que no caso, é apenas esperar 1000 milesegundos ( 1 segundo )

Testar o jogo, deve funcionar normalmente

Manipulando as mensagens

No arquivo ./src/tela.js, antes de tudo, inclua:

...

const util = Util;

const ID\_CONTEUDO = 'conteudo';

...

No arquivo ./src/Tela.js usamos apenas métodos estáticos, static, porque não é lugar para lógicas propriamente ditas, e static não trabalha com o contexto ‘this’, por isso não podemos passar a classe ./src/Util.js no construtor desta classe Tela, assim, o invocamos normalmente

Ainda em ./src/Tela.js, no método exibirMensagem, incluímos o async:

...

static async exibirMensagem(sucesso = true){

...

E na mesma função, mudamos:

...

    }

    elemento.classList.remove(CLASSE\_INVISIBLE)

}

...

Para:

...

    }

    elemento.classList.remove(CLASSE\_INVISIBLE)

    await util.timeOut(1000);

    elemento.classList.add(CLASSE\_INVISIBLE);

}

Retirando a CLASSE\_INVISIBLE, tornamos as mensagens visíveis

Usamos ‘util’ para aguardar 1 segundo

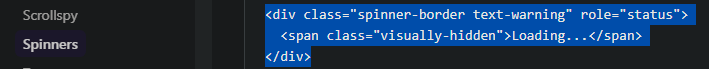
Incluímos a CLASSE\_INVISIBLE, para as mensagens ficarem novamente invisíveis



Incluindo um Spinner

Pequeno círculo loading

No site do BootStrap:



...

<body class="container-fluid">

    <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>

    <div id="carregando" class="row container-fluid invisible">

        <strong id="contador">Começando em 1 ...</strong>

        <div class="spinner-border text-warning spinner-border-sm" role="status">

            <span class="visually-hidden">Loading...</span>

        </div>

    </div>

    <br />

...

Coloque invisible

Tire o <br /> daqui e colocar aqui

Coloque esta linha

Coloque spinner-border-sm para diminuir o spinner

Loading nem vai aparecer??

Incluir

No método embaralhar:

...

        this.tela.exibirCarregando()

        await this.util.timeOut(1000)

        this.esconderHerois(copias)

        this.tela.exibirCarregando(false)

..

No arquivo ./src/Tela.js

Incluir estes 2 Ids:

...

…

Testar o jogo, deve funcionar normalmente

Incluir Contador

No arquivo ./src/Tela.js

...

static iniciarContador(){

let contarAte = 3;

const elementoContador = document.getElementById(ID\_CONTADOR)

    const identificadorNoTexto = "$$contador"

    const textoPadrao = `Começando em ${identificadorNoTexto} segundos...`

    const atualizarTexto = () =>

        (elementoContador.innerHTML = textoPadrao.replace(identificadorNoTexto, contarAte--))

    atualizarTexto();

    const idDoIntervalo = setInterval(atualizarTexto, 1000)

    return idDoIntervalo

}

...

let contarAte = 3; -> Inicializa a variável que conterá o valor do contador, 3, 2, 1...

const elementoContador = document.getElementById(ID\_CONTADOR) -> Identifica o objeto html onde será exibido a variável contarAté

const identificadorNoTexto = "$$contador" -> Criar uma variável que conterá um texto no meio de uma frase, depois, apenas este texto será alterado com o valor de contarAte, e não o texto todo

const textoPadrao = `Começando em ${identificadorNoTexto} segundos...` -> criando um texto padrão pra o contador em si, reparem que no meio do texto está o identificadorNoTexto, visto anteriormente

const atualizarTexto = () =>

(elementoContador.innerHTML = textoPadrao.replace(identificadorNoTexto, contarAte--)) -> Cria uma função que pega o conteúdo do texto padrão, procura nele o ‘texto’ $$contador e o sobrescreve, replace, com o valor de contarAte, e a cada chamada desta função a variável contarAté é decrementada em 1 – Observe que estamos criando uma função dentro de outra função, a iniciarContador()

atualizarTexto(); -> Já, aqui mesmo, dentro da função iniciarContador(), chamamos pela primeira e única vez a função atualizarTexto() que também está dentro de iniciarContador() com valor inicial de contarAte, que de cara é 3

const idDoIntervalo = setInterval(atualizarTexto, 1000) -> configuramos um setInterval(...) em uma âncora (identificador, variável idDoIntervalo) chamando a função anterior atualizarTexto() a cada segundo, 3, 2, 1, 0, -1, -2, o certo é 3, 2, 1 por isso temos que formular uma forma de parar o setInterval(...)

return idDoIntervalo -> Return da função iniciarContador() – O identificador da setInterval(...) anterior para que seja possível usá-lo para encerrar o setInterval(...)

No arquivo ./src/JogoDaMemoria.js, no método embaralhar()

...

this.tela.exibirCarregando()

const idDoIntervalo = this.tela.iniciarContador()

await this.util.timeOut(1000)

...

Se testar agora o nosso jogo, a cada clique no botão ‘Clique aqui para iniciar’ o contador irá aparecer mas os valores, os segundo em sí, estarão instáveis, as vezes mostrando números negativos, ect, simplesmente porque iniciamos o setInterval(...) lá em ./src/Tela.js no método iniciarContador()

Para resolver este problema, no arquivo .src/Tela.js no final do código vamos criar uma função para parar o contador

...

static limparContador(idDoIntervalo){

    clearInterval(idDoIntervalo)

    document.getElementById(ID\_CONTADOR).innerHTML = ""

}

...

clearInterval(idDoIntervalo) -> Desativa o setInterval(...) armazenado no idDoIntervalo

document.getElementById(ID\_CONTADOR).innerHTML = "" -> Limpa o conteúdo do identificadorNoTexto ( contarAte ) e todo o textoPadrao

No arquivo, ./src/JogoDaMemoria.js, no método embaralhar()

...

this.tela.exibirCarregando()

const idDoIntervalo = this.tela.iniciarContador()

await this.util.timeOut(1000)

this.tela.limparContador(idDoIntervalo)

…

Trocamos:

await this.util.timeOut(1000)

por:

await this.util.timeOut(3000)

Assim, o Contador junto com o texto padrão ficará visível apenas por 3 segundos, tempo suficiente para o this.tela.iniciarContador( ) fazer um decremento de 3 a 1

this.tela.limparContador(idDoIntervalo)

Esta função faz o setInterval(...) de this.tela.iniciarContador() parar a contagem

Testar o jogo, deve funcionar normalmente, o contador mudar de 3 para 2 para 1 e em seguida todo o texto do contador deixará de aparecer