Imersão Dev - Aula 08

Rafaella: Olá, **Devs**! Boas-vindas ao nosso oitavo dia de **Imersão Dev**. Estou muito feliz com os resultados que vocês têm postado nas redes sociais e vendo o quando vocês estão se superando a cada dia.

Hoje teremos mais um projeto incrível para vocês colocarem no portfólio e também vamos aprofundar um pouco mais na lógica de programação. Inclusive, não esqueça de postar o seu projeto no Linkedin, nos canais do Discord, no Instagram e onde mais você achar que as pessoas poderão ver. Sinta orgulho de tudo que você tem desenvolvido.

Pegue o seu café, abra o **CodePen** e vamos para a oitava aula!

Vamos dar uma relembrada na nossa aula anterior que eu vi que o Paulo Silveira está com algumas dúvidas. Ele gostaria de relembrar algumas coisas. Paulo, o que você gostaria de ver?

Paulo: Pois é, Rafa, pela primeira estamos fazendo um projeto um pouco maior, que vai envolver mais de uma aula. Serão duas aulas desse projeto para deixarmos bem redondo. É interessante reler o código que fizemos na aula passada e passar por alguns pontos nos quais talvez alguém possa ter uma dificuldade e que possa ter mais de uma opção de realizar o mesmo código.

O Guilherme está mostrando o código da **aula 7**, que vamos usar para a **aula 8**. E as primeiras linhas do código são as cartas que estamos fazendo igual e com chaves. Não é que dentro das chaves vai ter uma lista, mas vai ter pares formados por chave e valor, por exemplo, `nome: "Bulbasauro"` (nome e do lado vem uma string).

E até demos um passo a mais. Colocamos entre aspas um objeto dentro de um objeto, porque toda carta tem nome e toda carta tem atributos. Mas atributos não são do tipo string, atributos têm dentro de si chave e valor. No nosso caso, as chaves `ataque`, `defesa` e `magia` para cada uma das cartas.

E, no final, criamos na linha 28 `var cartas = [carta1, carta2, carta3];`, colocamos referências para as cartas (os objetos) dentro de uma lista, dentro de uma array. Repare aqui o uso dos colchetes para criar essa array. Tanto que aqui poderíamos até acessar cartas na posição zero “[0].atributos.ataque”. Para navegar nesse monte de objetos usa-se ponto e também aquela ferramenta de acessar os elementos do índice, os colchetes (`[]`) para acessar o enésimo elemento ou o primeiro, o segundo ou terceiro elemento.

Como criamos essa variável fora de função, todas as funções podem acessá-la. Chamamos isso de “escopo”. O escopo dessa variável está praticamente global, todo mundo pode acessar.

Em seguida, criamos duas outras variáveis, `cartaMaquina` e `cartaJogador`. Criaram fora também. Reparem que isso não foi sem querer, foi proposital. Para que as outras funções possam também acessá-las. E, logo abaixo, `sortearCarta()` é a função que tem o tal do *listener*. Você clica e ele dispara um evento que vai cair exatamente nessa

linha 32, `function sortearCarta()`. Essa função só será executada quando nós clicarmos.

Lembra que quando se cria uma função aquele trecho de código, se eu clicar em "Run", ele não roda sozinho. Alguém tem que mandar "sortear carta". Isso está amarrado no HTML.

Dentro de `sortearCarta()` sorteia a primeira carta, de 0, 1 ou 2 e pegamos a carta da máquina. Depois, para pegar a carta do jogador usamos um truque, que até deixamos um desafio para fazer de outras formas. Nós usamos esse laço, esse *loop* que parece o `for`, mas chama-se `while`. Porque não queremos sortear a mesma carta que a máquina. Então falamos: "Enquanto sorteou a mesma, continue sorteando". Eu quero que, obrigatoriamente, o número da carta da máquina seja diferente. Enquanto for igual, continua.

E, por fim, sorteia a carta do jogador. Então temos tanto a carta do jogador quanto a carta da máquina na mão. Em seguida, desabilitamos o botão de sortear e habilitamos o botão "Jogar". Além disso, chamamos uma função ali na linha 46, `exibirOpcoes()`, porque na hora em que abrimos o botão "Jogar", queremos que o usuário escolha com qual dos atributos do Super Trunfo quer jogar. Qual atributo da sua carta você quer usar?

Esse `exibirOpcoes()` envolve um pouco mais de HTML. Pegamos o elemento `opcoes` e começamos a preencher HTML, `"<input type='radio' name='atributo' value='"`, aí envolve mais HTML não é tanto JavaScript.

Para cada um dos atributos que uma carta tem - , ataque, defesa, magia, pode vir a ter outros - nós colocamos um *input*, colocamos botões radio. Quando terminou de exibir opções, o nosso programa para e fica esperando a pessoa clicar em "Jogar", selecionar uma opção e jogar.

Quando clica em jogar, podemos ver que tem uma outra função, mais para baixo, na linha 73, chamada `jogar()` que só vai ser executada quando alguém clicar em "Jogar". É uma função, ela só é chamada e disparada quando alguém clicar.

E o `jogar()` obtém o atributo selecionado para saber se foi ataque, defesa ou magia; pega o `elementoResultado` que é aquele espaço no HTML onde vai aparecer o resultado. Em seguida, fazemos a verificação. Pegamos o valor do atributo dele - se é ataque, quanto vale o ataque do jogador, quanto vale o ataque da máquina e compara.

Se o do jogador for maior, aparece "Você venceu", se for menor "Você perdeu". - é comum aparecer esse zigue-zague de `if`s - E, em último caso, aparece "Empatou" e trocamos o `innerHTML`, colocamos a informação na tela. Dá para colocar mais informações aqui, por exemplo, empatou por quanto, qual era a carta, etc. No final inserimos o `console.log()` só para vermos internamente.

O que estamos fazendo aqui é tentar navegar passo a passo, linha por linha, e acompanhar o que acontece no código quando clicamos. Isso é muito comum, em uma das aulas passadas o Guilherme chamou isso de "teste de mesa", podemos chamar também de "simular" ou até "debuggar".

Às vezes adquirimos tanta confiança que paramos de fazer isso, infelizmente. Fazer essa simulação é bom para entender o que está acontecendo, para ver se não tem nenhum bug e também para exercitar, entender comandos novos, encontrar e solucionar dúvidas. É bom para recapitular. Sugiro, inclusive, que você faça isso com outros códigos que você já escreveu. Recomendo pegar esse hábito de ler o código e narrar o que está acontecendo.

Rafaella: Concordo totalmente. Agora, daremos início à aula 8. Vamos pegar uma parte do código da aula 7 que já fizemos. Nele tem aquela função de jogar, de exibir o resultado, coisas que já tínhamos visto.

Porém, nesta aula vai ter algo muito legal. Vamos mostrar as cartas na tela. Tanto a carta do jogador quanto a carta da máquina. Para isso, já temos pronto o nosso código HTML e CSS. E o JavaScript que já vai vir pronto, aqui embaixo no CodePen da aula de hoje, como eu falei, é o código da aula 7. Então você vai poder se familiarizar tranquilamente por que é o que já tínhamos feito na aula anterior.

Gui, como vamos começar a incluir essas imagens nessa moldura? Para deixarmos o nosso projeto bem bonito.

Guilherme: Vamos lá. Eu vou fazer o *fork* desse projeto e também clicar no "Settings > Behavior" e deixar "não" nas opções de salvamento automático e de auto-update e deixar "sim" para a formatação. E no "Pen Details", vou escrever no campo de tags: "imersaodev, alura". O nome do projeto é "Super Trunfo - Dia 8" e, em seguida, vou clicar no botão "Salvar".

Paulo: Guilherme, a única diferença entre esse projeto e o projeto anterior é que vocês colocaram um pouco de HTML e CSS só para já preparar para colocarmos as cartas, é isso? O código JavaScript é o mesmo que aprendemos na aula passada e continuaremos do mesmo ponto.

Guilherme: Isso mesmo. A primeira coisa a fazer é criar uma nova propriedade para o nosso objeto chamado "imagem". Nela colocaremos o link da imagem que queremos utilizar. Vou começar com a carta do Paulo, `imagem:""` e dentro das aspas colocaremos o endereço da imagem que encontraremos na internet. Vou pesquisar por "Shiryu de Dragão" no Google, escolher a imagem. Para copiar o link dessa imagem vou clicar sobre ela com o botão direito e selecionar "Copiar endereço da imagem". Em seguida, vou colar esse endereço no nosso código entre as aspas de `imagem:""`.

```
var cartaPaulo = {
  nome: "Shiryu de dragão",
  imagem:

"http://pm1.narvii.com/6399/96fdb9d4fe6a9e72b9bc60ad418e3c43795e53b4_
00.jpg",
  atributos: {
    ataque: 5,
    defesa: 9,
```

```
    magia: 10
  }
};
```

Guilherme: E faremos a mesma coisa para as outras cartas, `cartaRafa` e `cartaGui`. Depois disso, temos que indicar de alguma forma no nosso código, depois do sorteio, que queremos exibir essa imagem da carta. Vamos por partes. Cuidaremos primeiro da carta do jogador e depois da carta da máquina.

Paulo: O código deve ser bem parecido, certo?

Guilherme: Sim. Vai mudar só alguns detalhes de um para o outro. Então, logo abaixo da linha 49, `exibirOpcoes()`, eu vou colocar `exibirCartaJogador()`.

Rafaella: Vamos fazer isso no final do sorteio, certo?

Guilherme: Isso. Depois de clicar no botão já poderíamos exibir a carta do jogador. Outra coisa, nós estávamos exibindo os atributos abaixo do botão "Jogar", mas queremos deixar os atributos aqui dentro da carta, logo abaixo da imagem, nós já preparamos um CSS para deixar isso bem bonito.

Como podemos fazer essa função, Rafa?

Rafaella: Primeiro vamos criar essa função com `function exibirCartaJogador()`.

Guilherme: Posso criá-la em qualquer lugar do código?

Rafaella: Pode. Acho que lá embaixo pode ser o ideal.

Guilherme: Antes disso, tem uma coisa legal que acabei não mostrando. Se eu clicar na numeração da lateral esquerda do nosso código ele agrupa o código de cada função, por exemplo, se eu clicar aqui no número 1 da primeira linha ele vai minimizar o código dessa função e vai ficar com o código minimizado entre as chaves `var cartaPaulo = {<>}`. Vou fazer isso em algumas funções para minimizar os trechos de códigos e deixar a tela mais limpa. É melhor para visualizar porque nosso projeto está ficando bem grande.

Então, agora, no fim do código, vou criar uma nova função.

```
function exibirCartaJogador(){

}
```

Rafaella: Perfeito. Teremos, como sempre temos no HTML, um id onde vamos colocar a nossa carta. Vocês já podem ver que a moldura das cartas está pronta na nossa página.

Vamos dar uma olhada no HTML, como está o id dessas cartas e pegar o id do jogador. Verificamos no HTML, `<div id="carta-jogador">`, que o id da `div` que queremos usar é `carta-jogador`. Vamos criar uma nova variável no JavaScript, uma `var divCartaJogador`. E vamos pegar com o `getElementById()` passando o id `carta-jogador`.

```
function exibirCartaJogador() {
  var divCartaJogador = document.getElementById("carta-jogador");
      }
```

Paulo: Esse id `carta-jogador` é o espaço que vocês criaram no HTML, que as pessoas que trabalham com design criaram, justamente para colocarmos a imagem lá dentro.

Rafaella: Exatamente. Isso já está pronto no HTML. Abrimos o HTML só para vocês que estão assistindo entenderem onde exatamente fica cada elemento. Depois teremos também o id `carta-maquina`. São duas `div`s diferentes, uma para a carta do jogador e outra para a carta da máquina.

Paulo: Lembrando que o nosso foco não está sendo em entender essa tag `<div>`, a tag `<img>`, a `<h3>`. Isso nós acabamos pegando por osmose. O importante aqui é a sequência que fazemos no código. Mas lembrando que nunca trabalhamos sozinhos. Trabalhamos em conjunto com a equipe de design e acho que amanhã isso deve ficar mais claro.

Rafaella: A próxima coisa a fazer, para exibir de fato as nossas imagens, é colocar as imagens dentro do nosso HTML, dentro desse elemento HTML. Como fazer isso? Você já sabe que conseguimos acessar as propriedades dessa tag colocando, por exemplo, `.value` para acessar o valor, já fizemos `.disabled` também, e para alterar o estilo da tag, que na verdade é o CSS dela, temos a propriedade `style` que também colocamos dentro do HTML.

Podemos ver, no código HTML, que temos a propriedade `style` na tag `<img>`, logo após a URL da imagem:

```
  <div id="carta-jogador">
   <img
src="https://www.alura.com.br/assets/img/imersoes/dev-2021/card-super
-trunfo-transparent-ajustado.png" style=" width: inherit; height:
inherit; position: absolute;">
   <h3></h3>
  </div>
```

Rafaella: Esse `style` é o próprio CSS, que escrevemos ali na outra aba do CodePen, mas dentro da própria linha do HTML. Existem vários motivos para fazer ou não fazer isso. Não vamos discutir agora sobre como escrever ou não o nosso CSS, mas é só

para vocês entenderem que esse `style` dentro da tag do HTML é o próprio CSS que você colocou dentro da tag. É como se ele estivesse aplicando apenas para essa tag `<img>`.

Então, vamos colocar a imagem dos nossos jogadores dentro do CSS dessa `div`. Para acessar essa propriedade usaremos o ponto. Temos essa variável `divCartaJogador`, que pegamos pelo id do elemento; `.style`, que é a propriedade do CSS; `.backgroundImage`, que é uma propriedade do CSS e usaremos para trocar a imagem. *Background image* é a imagem de fundo.

```
function exibirCartaJogador() {
  var divCartaJogador = document.getElementById("carta-jogador");
  divCartaJogador.style.backgroundImage =
}
```

Rafaella: Como já tínhamos feito anteriormente, para acessar uma propriedade dentro de uma propriedade, vamos colocando ponto e dessa forma é fácil entender o que estamos fazendo. Agora, vamos passar para o `divCartaJogador.style.backgroundImage` a URL que colocamos no atributo `imagem` dos nossos objetos no começo da aula.

Como passamos essa informação da URL dentro do CSS? Por mais que estejamos no código do JavaScript, precisamos fazer na notação que o CSS entenda. Para isso, vamos escrever com uma *template string*. É uma sintaxe um pouco diferente para podermos colocar JavaScript dentro do CSS, porque vamos pegar a variável que já colocamos no objeto, e ao mesmo tempo também precisamos escrever esse CSS.

Vai ser uma forma um pouco diferente de ver como podemos mesclar linguagens uma dentro da outra. Usaremos duas crases (``) e, dentro das crases, vamos inserir a informação da URL, `` `url ()` ``. Isso é notação do CSS, que escrevemos para poder colocar imagem. Dentro do parênteses passamos a URL que queremos acessar para a imagem aparecer no plano de fundo.

Mas, como eu falei, utilizaremos uma notação de *template string*. Então, colocaremos um cifrão (`$`) para indicar que o código a seguir é código de JavaScript e devemos colocar o código entre chaves. Colocaremos `{cartaJogador.imagem}`, que é o atributo no qual temos a URL da imagem dentro do nosso objeto.

```
function exibirCartaJogador() {
  var divCartaJogador = document.getElementById("carta-jogador");
  divCartaJogador.style.backgroundImage =
`url(${cartaJogador.imagem})`;
}
```

Então, essa será a notação. Não é muito intuitivo, mas, se analisarmos parte a parte, é possível entender tudo que está acontecendo. Estamos colocando o JavaScript dentro

da URL, que é a parte do CSS e alterando o `backgroundImage` da propriedade `style` da nossa *tag*, aquela estilização do nosso HTML.

Outra coisa que precisaremos alterar é a nossa moldura, porque queremos que apareça o nome do personagem. Para isso, faremos o `innerHTML`, assim, conseguiremos escrever na nossa *tag* `div`.

Paulo: Rafa, Guilherme, então, esse formato que assusta um pouco por conta das crases, na verdade, nós poderíamos também escrever com as aspas duplas, se usássemos vários sinais de mais "+". Vocês podem escrever na outra linha a mesma coisa, mas da forma antiga, que já sabíamos?

Eu compreendo que estamos mostrando esse jeito novo, porque ele é muito comum. Essa maneira de escrever uma *string* e, no meio, dizer que ela tem um pedaço de outra *string* é muito comum no JavaScript. Mas, também é possível escrever com vários "fecha aspas, mais, fecha aspas, aspas, fecha aspas". Escreva do outro jeito, só para compararmos, e, assim, compreendermos. Depois deixamos comentado.

Rafaella: Nós podemos colocar as aspas duplas, como utilizávamos antes. A parte da URL, deixamos como *string* mesmo. Tiraremos o cifrão e as chaves também, não precisaremos delas, já que são os indicadores de código JavaScript. Fecharemos as nossas aspas duplas da URL, colocaremos o sinal de mais "+" para adicionar a *string* à nossa variável, ao nosso código JavaScript e fecharemos os parênteses, também com aspas duplas.

```
function exibirCartaJogador() {
  var divCartaJogador = document.getElementById("carta-jogador");
  divCartaJogador.style.backgroundImage =
`url(${cartaJogador.imagem})`;
  // divCartaJogador.style.backgroundImage = "url(" +
cartaJogador.imagem + ")"
```

É o mesmo, porém, utilizamos mais a outra forma, porque, com o tempo, percebemos que com ela fica mais fácil entender o que está acontecendo e evitamos usar uma quantidade grande de sinais de mais e aspas.

Guilherme: Vou deixar essa linha comentada. Agora, se observarmos o HTML, nós temos a `div` da carta do jogador, e, dentro da `div`, uma imagem, com um *layout* definido, uma moldura de como as nossas cartas ficarão. Um detalhe importante é: na hora em que pedirmos para que o JavaScript monte a carta com a imagem que temos e o nome dela corretamente, precisaremos montar em uma ordem também correta.

Nós deixaremos o código da moldura na descrição do vídeo, para não ficarmos investindo tempo escrevendo HTML agora. Nós teremos aulas específicas para HTML, mas não agora. Queremos focar no JavaScript. Então, na descrição do vídeo, temos o

seguinte código da moldura, que eu também copiarei, também, da descrição e colarei no código. Basta fazer isso e teremos a moldura.

```
var moldura ='<img
src="https://www.alura.com.br/assets/img/imersoes/dev-2021/card-super
-trunfo-transparent.png" style=" width: inherit; height: inherit;
position: absolute;">';
```

Lembrando que a Rafa já pegou a `div` da carta do jogador, eu já peguei a variável com a moldura. Agora, precisaremos de outras opções, outras coisas que queremos mostrar na carta. Por exemplo, para mostrar o nome, a imagem da carta e as opções. Nós precisaremos montar essa estrutura com o JavaScript para que ela seja exibida no HTML.

Agora que já temos o *layout* da imagem que desejamos exibir, se observarmos a nossa carta no HTML, a carta do jogador, nós temos a imagem com o *layout* na tela. Portanto, são duas coisas diferentes. Nós queremos que a nossa, a imagem do Bulbassauro, do Vader ou do Shiryu de Dragão apareçam e continuar usando a nossa moldura. Para isso, precisaremos montar toda a estrutura no JavaScript.

Nós indicaremos que temos determinada imagem, que será usada como *background*, e precisaremos também da moldura. O código da moldura é HTML e está na descrição do vídeo. Vamos pegar o código HTML, copiar o código e colar embaixo.

```
 // divCartaJogador.style.backgroundImage = "url(" +
cartaJogador.imagem + ")"
  var moldura ='<img
src="https://www.alura.com.br/assets/img/imersoes/dev-2021/card-super
-trunfo-transparent.png" style=" width: inherit; height: inherit;
position: absolute;">';
```

Sendo assim, já temos a imagem que queremos usar com essa moldura. Depois, vamos apenas montá-la. Outro passo que podemos fazer, e que é muito importante, é exibir as opções. A opção, nós mostrávamos embaixo, montávamos embaixo do "sortear carta", não é isso que queremos nesse momento. Nós queremos exibir as opções como propriedades da carta.

Sendo assim, podemos montar, também, um trecho do HTML para criarmos essa `div` de opções para que a nossa função, que mostra as opções na tela, consiga incluir corretamente. Vamos fazer, Rafa?

Rafaella: A primeira coisa que faremos, para exibir as opções também, é criar uma *tag* para colocar as opções de fato. Portanto, criaremos uma var e podemos chamá-la de `tagHTML`, que será, exatamente, a parte das opções. E no nosso HTML, escreveremos,

entre aspas - assim como tínhamos na *tag* de cima - uma `div`. O `id` dela será “opções”, que colocaremos entre aspas simples, `var tagHTML = '<div id= 'opcoes' '`.

E colocaremos também uma classe, que será utilizada no CSS, assim como utilizamos no JavaScript o `id`, para direcionarmos qual elemento estamos usando, no CSS, às vezes também usamos a classe. Por isso, não se assustem, nós não trataremos disso agora, basta apenas vocês entenderem que nós utilizamos classes para a estilização também, que já está pronta.

Nós precisaremos indicar que a classe será o `carta-status`, ou seja, `var tagHTML = "<div id= 'opcoes' class='carta-status' "`.

Guilherme: Isso foi feito pelo pessoal que desenvolveu o *layout*. Nós só usaremos as propriedades para deixar bem bonito.

Rafaella: Sim! Também precisamos fechar a *tag* no final.

```
var tagHTML = "<div id= 'opcoes' class='carta-status'>".
```

Dentro dessa `tag` - dessa `div` - também colocaremos cada uma das nossas opções. Assim como havíamos feito na aula 7, inclusive, Gui, se você puder subir um pouco o código, apenas para relembrarmos. Na aula 7, criamos uma variável `opcoesTexto, varopcoesTexto = "";`, nós a inicializamos vazia e escrevemos, dentro dela, cada um dos *inputs*, de ataque, defesa e magia.

Nós faremos o mesmo, com a diferença de que antes ele aparecia na página grande, "jogado", e agora colocaremos os atributos dentro de cada uma das cartas, para deixarmos todos eles listados, como se realmente tivéssemos uma carta e pudéssemos escolher qual dos atributos queremos dela. O que você acha, Gui, de copiarmos e colarmos esse trecho de código?

```
var opcoes = document.getElementById("opcoes");
  var opcoesTexto = "";

  for (var atributo in cartaJogador.atributos) {
    opcoesTexto +=
      "<input type='radio' name='atributo' value='" +
      atributo +
      "'>" +
      atributo;
  }
```

Guilherme: Vamos!

Rafaella: Apenas para vocês perceberem que será exatamente o mesmo que fizemos na aula 7. Nós criamos a variável `opcoesTexto` vazia e fizemos uma estrutura de repetição do for, mas de uma maneira um pouco diferente, que utilizamos para listas. Vamos criando, para cada (for) elemento da lista, uma variável para chamar, do elemento, que será o nosso índice da estrutura de repetição.

Ou seja, para cada elemento (for) dentro (que é o in) e, então, vem a nossa lista, que, na verdade, não é bem uma lista, mas são alguns valores, algumas chaves, valores, nesse caso, porque o atributo era de fato, dentro do nosso objeto, era de fato algo que chamava outras chaves e valores. Para cada uma dessas coisas, estávamos escrevendo na tela.

Nós tínhamos nosso `opcoesTexto`, que seria a *tag*, o *input* do tipo `radio` - um botão que também estudamos e estávamos escrevendo o atributo e colocando o valor dele também. Agora, no nosso código, nós precisaremos indicar qual o valor do atributo.

Guilherme: Rafa, só para entender, você quer mostrar também o valor que aparece em cada atributo.

Rafaella: Isso! Que seria 10, 20, 30, a depender da força da magia que cada um dos personagens possuírem.

Guilherme: Combinado.

Rafaella: Nós adicionaremos, além do atributo que está na nossa *tag*, usando aspas e espaço, o valor do atributo. Assim, como também fizemos na aula anterior, para acessar ele valor, nós pegamos o objeto, no caso, `cartaJogador`, colocamos o `.atributos`, que é a primeira chave dentro do nosso objeto e depois os colchetes com atributo, de fato, que é o valor chave que desejamos.

```
var opcoesTexto = "";
  for (var atributo in cartaJogador.atributos) {
    opcoesTexto += "<input type='radio' name='atributo' value='" +
atributo + "'>" + atributo + "I" +
cartaJogador.atributos[atributo] ;
```

Se for, por exemplo, ataque, é como se estivéssemos colocando "ataque" dentro dos parênteses e quiséssemos retornar o valor desse “ataque”, porque estamos criando o var atributo que é uma nova variável para indicar cada um dos elementos dentro da nossa lista de atributos.

Dessa forma, podemos também colocar uma quebra de linha, para pularmos linha entre cada um desses *inputs* e, para isso, adicionamos também uma *tag* `<br>`, que representa uma quebra de linha dentro do nosso HTML.

```
  opcoesTexto += "<input type='radio' name='atributo' value='" +
atributo + "'>" + atributo + "I" +
cartaJogador.atributos[atributo] + "<br>";
```

Guilherme: Interessante, não é, Rafa, porque, no outro, ele aparecia em linha. Como queremos exibir na parte inferior da carta: "ataque", e, na frente, o valor; na linha abaixo, "magia" e, na frente, o valor da magia; na linha abaixo, "defesa" e, na frente, o valor da defesa.

Rafaella: Exato! Agora, precisamos também adicionar o nosso nome, que será o nome do personagem que aparecerá na parte superior da carta. Temos até um espaço reservado para isso.

Guilherme: Vai ficar bem bonito!

Rafaella: Para isso, nós criaremos uma nova variável var nome e colocaremos o nome de cada um dos personagens dentro de uma tag "p", que é uma *tag* de parágrafo. Podemos utilizar a notação de *template string*, que acabamos de estudar, assim, nos servirá também como treino. Portanto, adicionaremos a tag `` `<p>` `` e, dentro dela, passaremos algumas propriedades. A primeira delas será a `class`, "classe", utilizada no nosso CSS já pronto para estilização.

A `class` será "carta-subtitle", `var nome = `<p class="subtitle"``. Nós já sabemos isso, porque o CSS já está criado, isto é, estou falando para vocês que é isso, porque o CSS é estilizado dessa forma. E, para adicionarmos o nosso código JavaScript, nós usaremos cifrão e abriremos chaves, para indicar qual o nome de cada um dos nossos personagens.

Para acessar essa informação, colocamos `cartaJogador.nome` e fechamos a nossa *tag* com `</p>`.

```
var nome = `<p class="carta-subtitle">${cartaJogador.nome}</p>`;
```

Agora, falta apenas escrever na tela tudo que já vimos. Para isso, utilizaremos o `innerHTML`. Então, colocaremos `divCartaJogador`, que é o elemento que desejamos escrever, `.innerHTML` e, dentro desse HTML, nós passaremos, primeiro, a moldura, depois, soma, já que são *tags* HTML que vamos adicionando uma após a outra, continuando, escreveremos o nome, que é a nossa variável, assim como criamos mais acima, com a *tag* `<p>` pronta.

Também passaremos o `tagHTML`, que foi a *tag* criada para as opções e colocaremos o `opcoesTexto`, que é, de fato, cada um dos atributos que queremos escrever e ficarão

dentro da nossa `tagHTML`. Agora, somamos com um fechamento de `div`. O motivo é que nós abrimos a `div`, como vocês podem ver no `tagHTML`, mas não fechamos.

Nós colocamos, dentro dessa `div`, o `opcoesTexto`, justamente por isso, ele aparece logo em seguida. Precisamos fechar a nossa *tag* no final. Então, como não conseguimos ficar fechando para cada estrutura de repetição, quer dizer, nós não conseguimos colocar no final do `opcoesTexto` o fechamento dessa `div`, é melhor deixar dentro, porque pode acontecer de fechar várias vezes e teremos um problema sério.

Sendo assim, fecharemos agora a nossa `div`.

```
divCartaJogador.innerHTML = moldura + nome + tagHTML + opcoesTexto +
</div>";
}
```

Então, é isso! Vamos salvar, rodar e verificar se tudo que fizemos vai funcionar.

Paulo: Rafa, Guilherme, as pessoas estão perdoadas se quiserem copiar e colar só esse trecho de código, porque temos uma série de pontos e vírgulas, fecha aspas, abre aspas, mas, relembrando, parece complicado, porque é a primeira vez que estamos colocando muito HTML no JavaScript. Isto é, o JavaScript gerando o HTML. Realmente, pode dar uma assustada, mas é normal!

O que a Rafa e o Guilherme fizeram é: é necessário ter determinada imagem, determinadas opções, um nome em determinada parte da carta e, cada um, é um tipo de *tag*. Um é um parágrafo, outro é uma divisão de espaço, outro é uma imagem e outro é um *input*. E temos várias aspas, concatenações e interpolações de *string*, que acaba dificultando um pouco.

Além disso, muitos sinais de mais "+", aspas simples, aspas duplas, enfim, é muito fácil errar e tomar um susto. Mas, é assim que encaramos e aprendemos.

Guilherme: Só um detalhe, Rafa. Antes de rodarmos, nós colocamos as opções dentro da nossa função que exibe o jogador. Eu posso remover esse `exibirOpcoes();` agora, não?

Rafaella: Pode.

Paulo: Mas, ela poderia ficar também, porque, não tem ninguém chamando. O problema é que estávamos, sim, chamando.

Guilherme: Sim, estávamos chamando dentro do `sortearCarta()`.

Paulo: Nós devemos tirar, porque o `exibirCartaJogador();` já exibe as opções.

Guilherme: Isso mesmo! Continuando, vou executar, sortear a carta e sorteou o Shiryu de Dragão, mas ele está com a imagem quebrada.

Rafaella: Provavelmente essa imagem não ficou boa.

Guilherme: Isso! Se fosse a carta do Vader, repare que, no final, ela tem um `.jpg` e a Bulbassauro também, então, faz sentido colocarmos imagens que são, de fato, imagens.

Rafaella: Então, vamos rodar para ver o que vai acontecer. Para isso, precisamos salvar e rodar. Já apareceu! Temos o Darth Vader, ficou muito legal! Temos o nosso ataque, defesa e magia, cada qual com os valores que definimos. Acho que poderíamos testar se outro personagem aparece.

Guilherme: Está sorteando o Vader toda hora.

Paulo: O dado está viciado!

Rafaella: Como eu havia comentado, o Bulbassauro estava como `.png`, que é aquele tipo de imagem com fundo transparente. O ideal é, realmente, não utilizarmos esse tipo de imagem, porque ela pode quebrar. Então, é interessante sempre checarmos se quando pegamos uma imagem no Google, ou seja lá onde vocês forem buscar a imagem para usar no jogo, se ela tem o final em `.jpg`, ao invés de `.png` e tenha várias outras coisas escritas.

Então, é interessante tentar procurar por imagens que tenham endereço com `.jpg` mesmo.

Guilherme: Rafa, deixa só eu fazer um teste. Se eu copio esse caminho da imagem até o `.png` e colo, vamos ver o que aparecerá. Ele está quebrando! Essa imagem, por algum motivo, não está funcionando. Vamos trocar! Podemos começar com o Shiryu de Dragão?

Rafaella: Vamos!

Guilherme: Então, vamos pesquisar uma imagem no Google. Vou pedir para abrir em uma nova aba, checar o final dela. Essa tem o `.jpg`. Provavelmente, essa ficará linda! Então, vamos remover o endereço anterior e colocar o correto.

```
var cartaPaulo = {
  nome: "Shiryu de dragão",
  imagem:
```

```
"http://pm1.narvii.com/6399/96fdb9d4fe6a9e72b9bc60ad418e3c43795e53b4_
00.jpg",
  atributos: {
    ataque: 5,
    defesa: 9,
    magia: 10
  }
};
```

Falta só o Bulbassauro. Então, vamos pesquisar no Google "Bulbassauro". Em seguida, abrir uma imagem, copiar o endereço e colar em outra aba. Esse endereço está muito grande, está estranho!

Rafaella: É em `.png`, provavelmente.

Guilherme: Sim, o final dela é em `.png`. Parece que teremos problemas com ela, não é, Rafa?

Rafaella: Sim, é bom garantir que será `.jpg`.

Guilherme: Vou salvar, executar e clicar em "Sortear". Veio o Darth Vader novamente, o lado sombrio da força está pesado aqui. Vou sortear de novo, agora veio o Bulbassauro, com a imagem aparecendo na tela. Só falta o Shiryu.

Paulo: Lembrando que a gente pode forçar, ao invés de passar o `Math.random()` colocar a carta que a gente quiser. Ou então você pode fazer `cartaJogador = cartas[0]`, assim a gente consegue forçar a carta. Vai aparecer o Shiryu na tela, incluindo a imagem que adicionamos.

```
function sortearCarta() {
  var numeroCartaMaquina = parseInt(Math.random() * 3);
  cartaMaquina = cartas[numeroCartaMaquina];

  var numeroCartaJogador = parseInt(Math.random() * 3);
  while (numeroCartaJogador == numeroCartaMaquina) {
    numeroCartaJogador = parseInt(Math.random() * 3);
  }
  cartaJogador = cartas[0];
  console.log(cartaJogador);
//...
```

Guilherme: A maior magia do nosso baralho.

Paulo: Pois é, se chama Cólera do Dragão.

Guilherme: Vou voltar o código a como ele estava antes, com `cartaJogador = cartas[numeroCartaJogador]`. Nós vimos que as imagens estão corretas. Acho que vale a pena testar e verificar se a extensão das imagens está aparecendo, seja `.jpg` ou `.png`. Se tiver um endereço muito longo, ele pode quebrar.

Rafaella: Ou às vezes pode redirecionar para o site onde a imagem está, aí não vai exibir essa imagem.

Guilherme: Exato. Aqui acho que está legal, Rafa. Será que se a gente clicar em algum atributo e colocar para jogar já vai funcionar?

Rafaella: Vamos ver o que acontece.

Guilherme: Vou jogar com o Shiryu de Dragão, selecionar a magia e clicar em "Jogar". Porém, não vai acontecer nada. Acho que vale a pena verificarmos o que precisa mudar em nossa função `jogar()`.

Nós estamos obtendo o `atributoSelecionado` e fazendo uma verificação para saber se o valor do atributo é maior ou menor que o da máquina.

Paulo: A formatação do CodePen está estranha, né? Mas tudo bem.

Guilherme: No `jogar()`, repare que havíamos colocado uma tag para exibirmos o resultado. Porém, agora estamos com outro visual, e queremos mostrar mesmo esse resultado na tela. Existe uma `<div>` com id `resultado` em nosso HTML, e é com ela que vamos trabalhar.

Primeiramente, vou criar uma variável `divResultado` recebendo `document.getElementById("resultado")`.

Em nosso primeiro caso, se a carta do jogador for maior que a carta da máquina, queremos exibir uma mensagem informando que a pessoa venceu o jogo. Portanto, nesse `if`, atribuiremos à variável `divResultado` o valor `"<p class='resultado-final'>Venceu</p>"`, incluindo uma tag `<p>`, uma classe `resultado-final` que preparamos previamente e o nosso texto "Venceu".

Também teremos esse comportamento para os outros casos. Por exemplo, quando a carta do jogador for menor que a da máquina, nossa variável `divResultado` receberá `"<p class='resultado-final'>Perdeu</p>"`.

Por fim, em caso de empate, esta variável receberá `htmlResultado = "<p class='resultado-final'>Empatou</p>"`. Vamos testar dessa forma para verificarmos se os resultados estão aparecendo na tela.

```
function jogar() {
  console.log("chamou");
  var atributoSelecionado = obtemAtributoSelecionado();
  var divResultado = document.getElementById("resultado");

  if (
    cartaJogador.atributos[atributoSelecionado] >
    cartaMaquina.atributos[atributoSelecionado]
  ) {
      divResultado = "<p class='resultado-final'>Venceu</p>";
  } else if (
    cartaJogador.atributos[atributoSelecionado] <
    cartaMaquina.atributos[atributoSelecionado]
  ) {
      divResultado = "<p class='resultado-final'>Perdeu</p>";
  } else {
      divResultado = "<p class='resultado-final'>Empatou</p>";
  }
  divResultado.innerHTML = divResultado

  document.getElementById("btnJogar").disabled = true;
  exibirCartaMaquina();
}
```

Ao salvar, executar, selecionar um atributo e jogar, não vai aparecer nada.

Rafaella: Não aparece pois não estamos fazendo o `innerHTML`, não é?

Guilherme: Verdade Rafa, muito boa.

Rafaella: Ali embaixo, depois que terminamos as verificações com o `if`, vamos colocar `divResultado.innerHTML` recebendo o `divResultado`, com mesmo nome. Porém, ao salvarmos e executarmos, o botão de "Jogar" não vai funcionar.

Acho que está tendo algum conflito que não está deixando a gente jogar de fato, e acho que sei o que é. Repare que estamos criando uma variável `divResultado` que recebe o elemento pelo ID,e estamos usando o mesmo nome para criar nossas tags `<p>`. O ideal é criarmos outro nome para essas tags, por exemplo `htmlResultado`.

```
function jogar() {
  console.log("chamou");
  var atributoSelecionado = obtemAtributoSelecionado();
  var divResultado = document.getElementById("resultado");

  if (
    cartaJogador.atributos[atributoSelecionado] >
    cartaMaquina.atributos[atributoSelecionado]
  ) {
      htmlResultado = "<p class='resultado-final'>Venceu</p>";
  } else if (
    cartaJogador.atributos[atributoSelecionado] <
    cartaMaquina.atributos[atributoSelecionado]
  ) {
      htmlResultado = "<p class='resultado-final'>Perdeu</p>";
  } else {
      htmlResultado = "<p class='resultado-final'>Empatou</p>";
  }
  divResultado.innerHTML = htmlResultado;
}
```

Guilherme: Executando dessa forma, vai funcionar perfeitamente. Agora precisamos desabilitar o botão "Jogar".

Rafaella: Para desabilitarmos o botão, como vimos na última aula, podemos usar a propriedade `disabled`. Usaremos o `document.getElementById()` para recuperarmos o botão que desejamos desabilitar, nesse caso o `btnJogar`, e alteraremos a propriedade `disabled` para `true`.

```
function jogar() {
  console.log("chamou");
  var atributoSelecionado = obtemAtributoSelecionado();
  var divResultado = document.getElementById("resultado");

  if (
      cartaJogador.atributos[atributoSelecionado] >
    cartaMaquina.atributos[atributoSelecionado]
  ) {
      htmlResultado = "<p class='resultado-final'>Venceu</p>";
  } else if (
    cartaJogador.atributos[atributoSelecionado] <
    cartaMaquina.atributos[atributoSelecionado]
```

```
  ) {
      htmlResultado = "<p class='resultado-final'>Perdeu</p>";
  } else {
      htmlResultado = "<p class='resultado-final'>Empatou</p>";
  }
  divResultado.innerHTML = htmlResultado;

  document.getElementById("btnJogar").disabled = true;
}
```

Guilherme: Beleza, vou salvar e executar. Ao clicar em "Jogar", a mensagem "Venceu" é exibida e o botão é desabilitado.

Rafaella: Falta exibirmos a carta da máquina, mas ela tem umas coisas diferentes - por exemplo, não precisamos colocar o `radio button`, basta escrevermos os atributos e seus valores. Como podemos fazer essa parte, Gui? Eu sei que muita coisa podemos pegar do `exibirCartaJogador`.

Guilherme: Acho que faz sentido. Na própria função jogar(), como última instrução, vou chamar a função `exibirCartaMaquina()`, que vamos criar agora.

Paulo: Está criando uma nova função apenas por organização, certo? Porque poderia colocar aí mesmo o código para exibir a carta da máquina.

Guilherme: Isso. Na função `exibirCartaJogador()` temos toda a estrutura que precisamos para montar a carta, e se observarmos no HTML já temos uma `<div>` para a carta da máquina, que tem propriedades bem parecidas com a carta do jogador. A única diferença é que não queremos os radio buttons, podemos só escrever os atributos.

Vou criar a função `exibirCartaMaquina()`. Vou copiar e colar o código de `exibirCartaJogador`, para depois alterar o trecho em que trabalhamos com a tag radio button.

Alteraremos a variável `divCartaJogador` para `divCartaMaquina` e a propriedade que estamos acessando com o `getElementById()` para `carta-maquina`. Também alteraremos as informações da imagem que é exibida, usando `divCartaMaquina.style.backgroundImage` e `url(${cartaMaquina.imagem}`.

Rafaella: A moldura continua a mesma e as opções também, mas não precisaremos mais do `input` do tipo `radio`. Ao invés dele, colocaremos uma tag `<p>` com o tipo `text`.

Também é possível apenas remover, nesse caso o `type` não é necessário. E ao final fecharemos essa tag com um `</p>`.

Por fim, alteraremos as outras referências de `cartaJogador` para `cartaMaquina`.

Paulo: Aqui já fica aquele alerta para quando você copiar e colar código. Muitas vezes você pode esquecer de mudar alguns valores e variáveis, o que vai trazer problemas na execução. É necessário tomar cuidado.

```
function exibirCartaMaquina() {
  var divCartaMaquina = document.getElementById("carta-maquina");
  divCartaMaquina.style.backgroundImage =
url(${cartaMaquina.imagem});
  // divCartaJogador.style.backgroundImage = "url(" +
cartaJogador.imagem + ")"
  var moldura =
      '<img
src="https://www.alura.com.br/assets/img/imersoes/dev-2021/card-super
-trunfo-transparent.png" style=" width: inherit; height: inherit;
position: absolute;">';
  var tagHTML = "<div id='opcoes' class='carta-status'>";

  var opcoesTexto = "";
  for (var atributo in cartaMaquina.atributos) {
      opcoesTexto +=
      "<p type='text' name='atributo' value='" +
      atributo +
      "'>" +
      atributo +
      " " +
      cartaMaquina.atributos[atributo] +
      "</p><br>";
  }
  var nome = <p class="carta-subtitle">${cartaMaquina.nome}</p>;

  divCartaMaquina.innerHTML = moldura + nome + tagHTML + opcoesTexto
+ "</div>";
}
```

Rafaella: Vamos salvar e rodar para vermos como ficou.

Guilherme: Sorteei a carta, saiu o Shiryu, vou selecionar o ataque. Ao clicar em "Jogar", aparece a mensagem "Perdeu" na tela e a carta do "Bulbassauro". Aconteceu alguma coisa que a carta da máquina ficou com os textos espaçados, e não está exibindo todos os valores.

Paulo: É o `<br>` ali na função `exibirCartaMaquina`. Pode tirar.

```
function exibirCartaMaquina() {
  var divCartaMaquina = document.getElementById("carta-maquina");
  divCartaMaquina.style.backgroundImage =
url(${cartaMaquina.imagem});
  // divCartaJogador.style.backgroundImage = "url(" +
cartaJogador.imagem + ")"
  var moldura =
      '<img
src="https://www.alura.com.br/assets/img/imersoes/dev-2021/card-super
-trunfo-transparent.png" style=" width: inherit; height: inherit;
position: absolute;">';
  var tagHTML = "<div id='opcoes' class='carta-status'>";

  var opcoesTexto = "";
  for (var atributo in cartaMaquina.atributos) {
      opcoesTexto +=
      "<p type='text' name='atributo' value='" +
      atributo +
      "'>" +
      atributo +
      " " +
      cartaMaquina.atributos[atributo] +
      "</p>";
  }
  var nome = <p class="carta-subtitle">${cartaMaquina.nome}</p>;

  divCartaMaquina.innerHTML = moldura + nome + tagHTML + opcoesTexto
+ "</div>";
}
```

Guilherme: Vamos ver, vou salvar e executar. Ao jogar novamente, agora a carta da máquina também é exibida com os valores corretos.

Rafaella: É que o `<br>` é a quebra de linha, e como o parágrafo já quebra, e esquecemos de tirar, gerou esse problema. Agora ficou maravilhoso.

Quais são nossos desafios para essa aula? Eu já tenho um, muito interessante, que seria adicionar um baralho com várias outras cartas - e o número vocês podem escolher, podem ser 3, 10, 21 e assim por diante.

Acho que vocês também poderiam criar um sistema onde, ao vencer, você ganha a carta do outro jogador. Isso é uma coisa mais complexa, que vai exigir bastante lógica e cuidado na hora de testar, mas seria muito legal. É justamente isso que acontece no super trunfo, quando ganhamos nós ficamos com a carta do outro jogador.

Paulo: Então ao invés de termos um array, teríamos dois arrays de cartas, é isso?

Rafaella: Exatamente.

Paulo: Teríamos as cartas do jogador, as cartas da máquina, e a cada round jogaríamos uma. Quem ganhar pega essa carta, remove de um array e coloca na outra para que ela possa ser sorteada. Por exemplo, nesse cenário nós perdermos, a carta iria para a máquina e nosso array ficaria menor. Quem chegar a 0 itens no array (com o valor `0` de `length`) perde.

Realmente é difícil, mas acho que dá para fazer. É mais código e menos HTML, então dá para encarar, gosto muito dessa ideia da Rafa. Para deixar o jogo mais justo, poderíamos até deixar a carta selecionar o atributo em algumas rodadas. Tem muita variação para isso aqui realmente virar um jogo.

Rafaella: Gui, você tem um desafio também?

Guilherme: Eu tenho um, acho que não tão complexo. Se observarmos nossas funções `exibirCartaMaquina()` e `exibirCartaJogador()`, elas são bem parecidas e alguns trechos mudam. Existe uma maneira de criarmos uma função `exibirCarta()` que funcione tanto para a máquina quanto para o jogador, dependendo de alguns parâmetros que forem passados para ela.

Meu desafio é vocês estudarem um pouco sobre isso, o que é passagem de parâmetros, como fazer uma função realizar dois comportamentos e assim por diante.

Paulo: Eu não vou dar desafio porque tem bastante coisa, só queria relembrar porque nessa aula nós avançamos muito. O código, se você reparar, está até mais simples do que aquele que fizemos na outra aula, mas temos muito HTML misturado, com o `innterHTML`, criação de `<div>`, `<p>`, tudo isso para mostrar as informações na tela.

Eu gostaria que vocês não ficassem apavorados ou apavoradas, tomem seu tempo, conversem com a gente e tirem dúvidas caso o código de vocês não funcione.

Esse projeto que o Gui e a Rafa elaboraram ficou muito legal. Aqui você pode trocar a moldura da carta, pode fazer os decks dos seus personagens favoritos, fazer os cards

dos instrutores e instrutoras do Aluraverso, incluir uma animação ou um áudio, muita coisa legal para dar seu toque pessoal.

Assim você consegue mostrar para as pessoas e ter um portfólio personalizado, diferente daqueles de outras pessoas, e incluir no seu Linkedin, nas redes sociais e assim por diante. Veja que realmente fechamos, em duas aulas, um projeto mais elaborado. Amanhã vamos para outro projeto para finalizarmos a imersão com chave de ouro, algo que vai nos deixar ainda mais orgulhosos de ver você colocar no ar.

Até amanhã!

Rafaella: Até amanhã!

Guilherme: Tchau, tchau!