

React

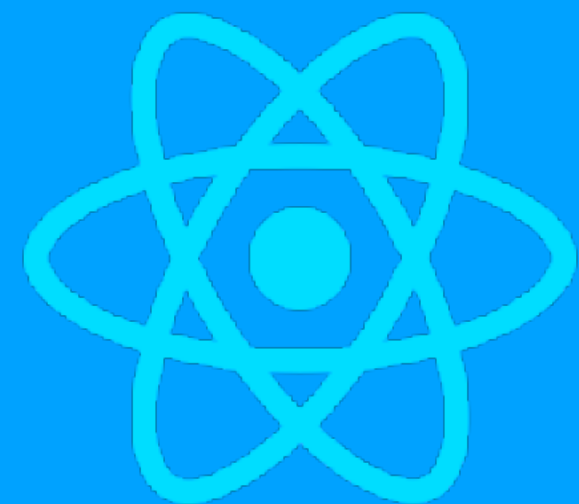
**Desenvolvendo um jogo com
ReactJS**

Parte 1

Por dentro do React - Uma breve introdução sobre esse excelente framework



ReactJS é uma biblioteca Javascript que foi criada pelo time do **Facebook** com o objetivo de **facilitar a criação e o desenvolvimento** de componentes para interfaces web;

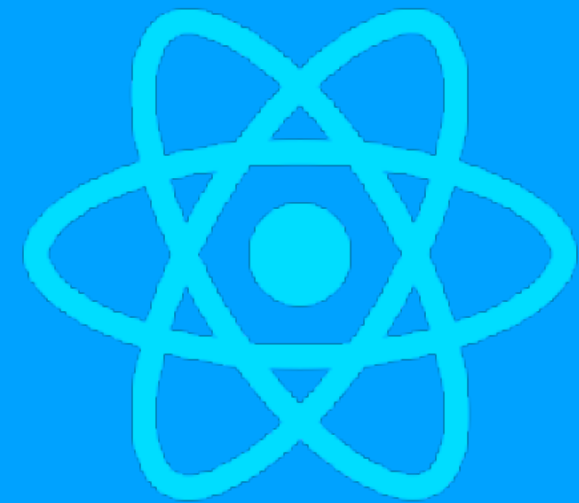


Simples: ReactJS é simples e ele automaticamente administra todas as atualizações da sua interface, atualizando-a quando seus dados mudam;

Baseado em componentes: desenvolvimento baseado em componentes, que facilitam a construção de UIs (User Interfaces)

Multiplataforma: Uma vez escrito o seu código você poderá rodá-lo em plataformas mobile (React-Native) ou mesmo em um servidor NodeJS.

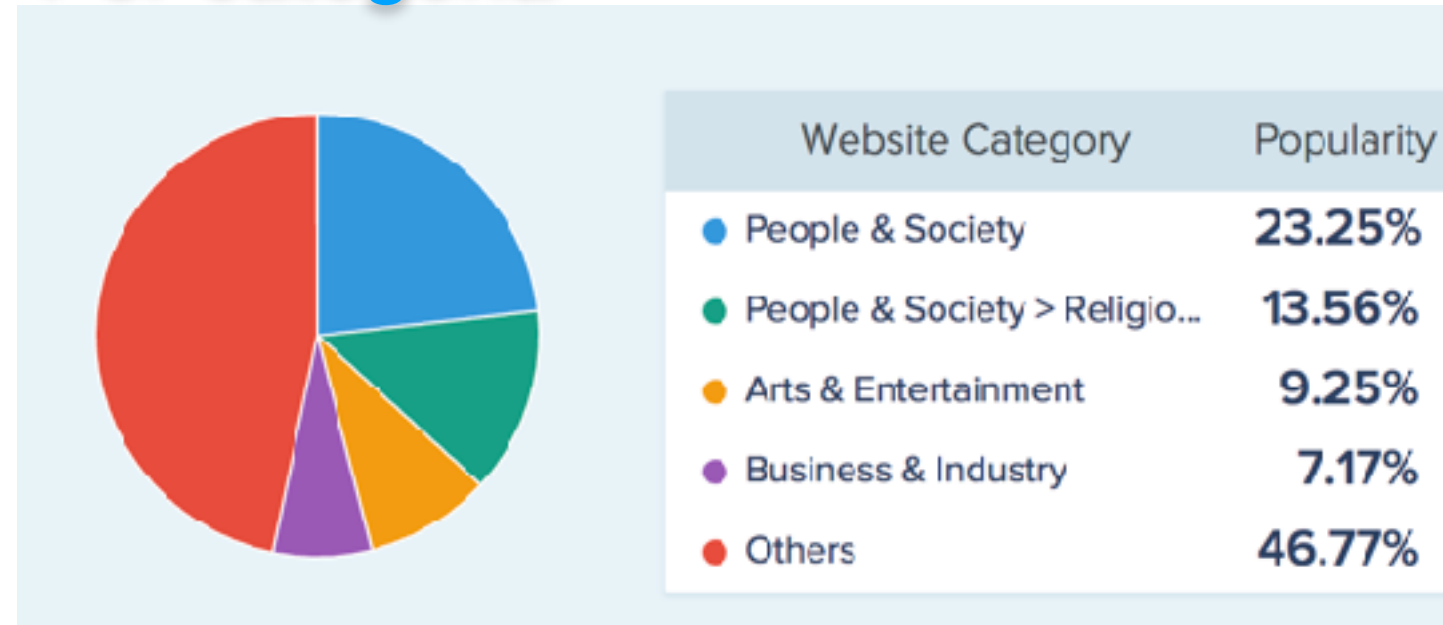
TIBCO - 2018



Feb 2018	Feb 2017	Change	Programming Language	Ratings	Change
1	1		Java	14.988%	-1.69%
2	2		C	11.857%	+3.41%
3	3		C++	5.726%	+0.30%
4	5	▲	Python	5.168%	+1.12%
5	4	▼	C#	4.453%	-0.45%
6	8	▲	Visual Basic .NET	4.072%	+1.25%
7	6	▼	PHP	3.420%	+0.35%
8	7	▼	JavaScript	3.165%	+0.29%
9	9		Delphi/Object Pascal	2.589%	+0.11%
10	11	▲	Ruby	2.534%	+0.38%
11	-	▲	SQL	2.356%	+2.36%
12	16	▲	Visual Basic	2.177%	+0.30%
13	15	▲	R	2.086%	+0.16%
14	18	▲	PL/SQL	1.877%	+0.33%
15	13	▼	Assembly language	1.833%	-0.27%
16	12	▼	Swift	1.794%	-0.33%
17	10	▼	Perl	1.759%	-0.41%
18	14	▼	Go	1.417%	-0.69%
19	17	▼	MATLAB	1.228%	-0.49%
20	19	▼	Objective-C	1.130%	-0.41%











Estatísticas de Uso

Por categoria

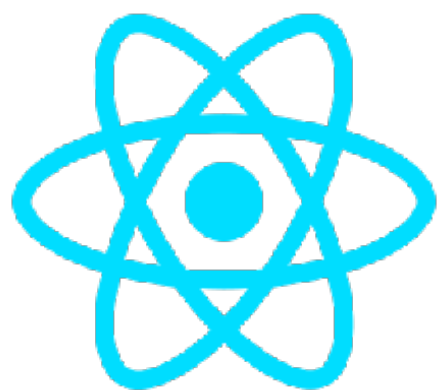


**605.158
apps**

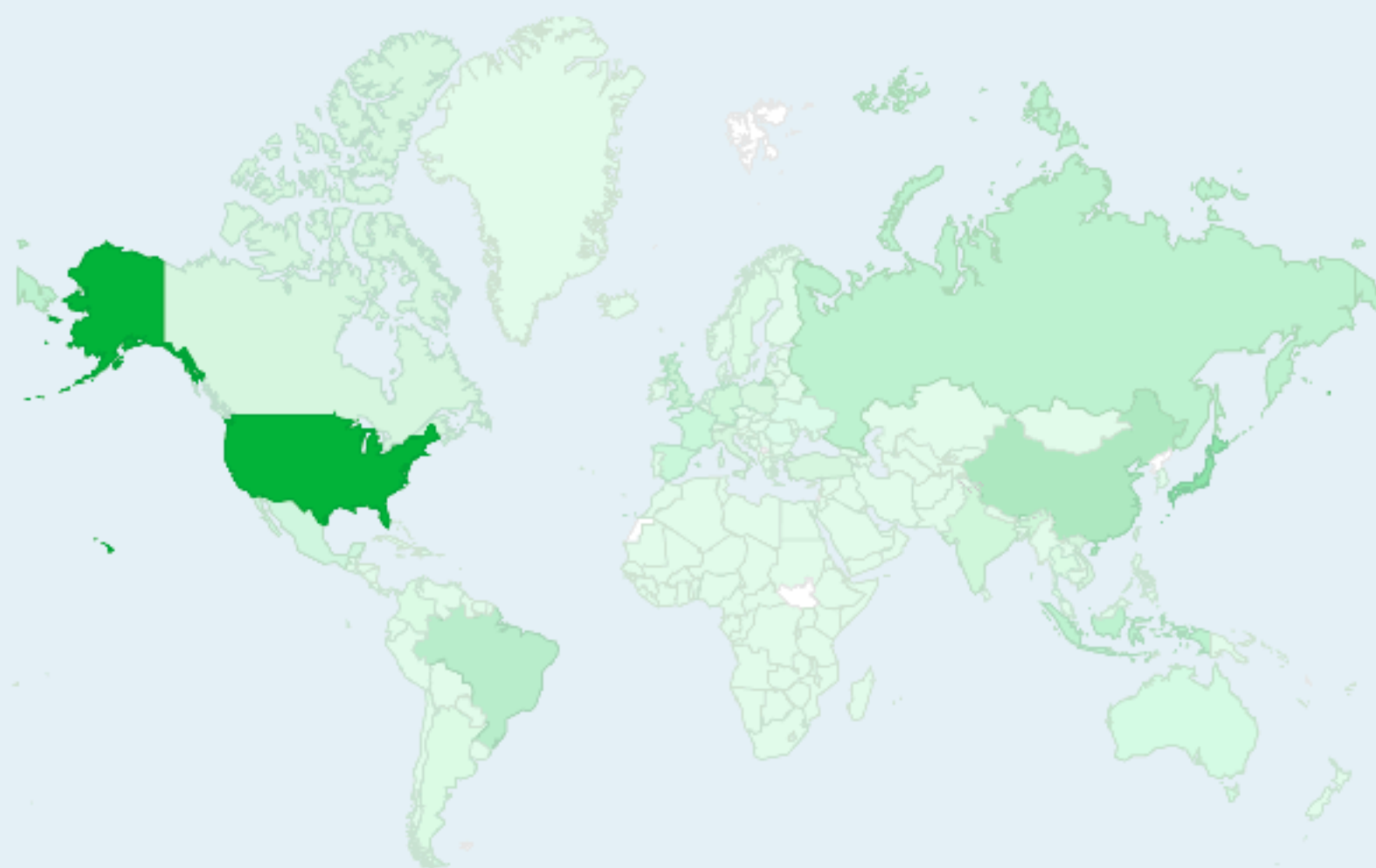
Top websites

 yahoo.com	9 TRAFFIC RANK	5.2B MONTHLY VISITS
 bing.com	46 TRAFFIC RANK	1.3B MONTHLY VISITS
 pinterest.com	55 TRAFFIC RANK	943.1M MONTHLY VISITS
 zhihu.com	95 TRAFFIC RANK	911.5M MONTHLY VISITS
 microsoft.com	79 TRAFFIC RANK	887.7M MONTHLY VISITS
 imdb.com	69 TRAFFIC RANK	773.4M MONTHLY VISITS
 imgur.com	75 TRAFFIC RANK	694.8M MONTHLY VISITS
 bbc.co.uk	106 TRAFFIC RANK	689M MONTHLY VISITS
 paypal.com	72 TRAFFIC RANK	628.8M MONTHLY VISITS
 mercadolibre.com.br	88 TRAFFIC RANK	501.9M MONTHLY VISITS

**140.236
Domínios no mundo**



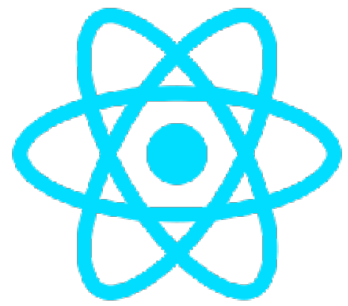
Países que lideram o uso



1 50,762

Leading Countries

Country	Websites
 United States	50,762
 Japan	18,521
 China	12,713
 Brazil	9,620
 Russia	8,232
 Indonesia	7,954
 United Kingdom	6,058
 France	5,035
 Spain	4,997
 Germany	4,614
Rest of the World	62,960

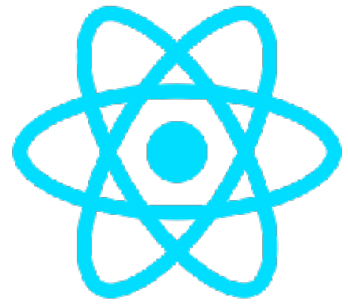


Entendo o React - Função Render

```
9  <body>
10  <div id="example"></div>
11  <script type="text/babel">
12    ReactDOM.render(
13      <h1>Hello, world!</h1>,
14      document.getElementById('example')
15    );
16  </script>
17 </body>
```

O método render é um dos métodos mais importantes do React e que será responsável por renderizar os elementos. Ele recebe 3 parâmetros, que são:

1. O elemento a ser criado. Veja o **<h1> Hello, World!</h1>**
2. O local onde será inserido o DOM e uma função de callback (retorno), que será chamada logo após a renderização.



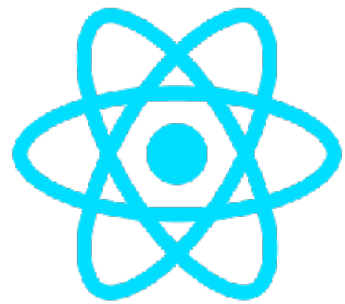
Entendo o React - JSX

O JSX (**J**ava**S**cript **X**ML) é um poderoso, porém às vezes controverso recurso do React. Com ele, podemos misturar tags HTML com código JavaScript

```
ReactDOM.render(  
  <h1>Hello World!</h1>,  
  document.getElementById("root")  
)
```

Agora veja o mesmo exemplo sem JSX

```
ReactDOM.render(  
  React.createElement('h1', null, "Hello World!"),  
  document.getElementById("root")  
)
```



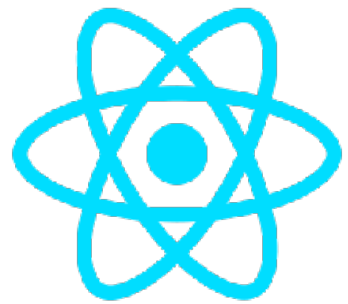
Entendo o React - React.createClass

ReactClass.createClass(object specification)

```
var Hello = React.createClass({  
  render: function() {  
    return (  
      <h1>Hello World!</h1>  
    );  
  }  
});  
ReactDOM.render(  
  <Hello />,  
  document.getElementById("content")  
);
```

Serve para criar um componente dada uma especificação. A vantagem do método é que nos permite criar componentes para serem reaproveitados.

Como podemos notar, com a utilização do **React.createClass**, habilitamos um componente com o nome da variável determinada, no nosso caso **Hello** e então podemos chamá-lo em diversos lugares como **<Hello />**

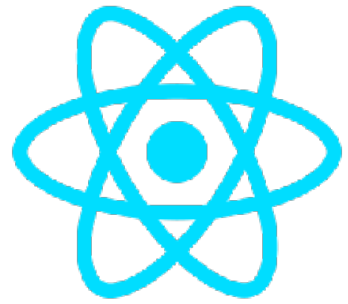


Entendo o React - React.componente

No ES6 (versão moderna do JavaScript), nós ganhamos vários recursos importantes e um deles são as Classes. Com ele é possível extender métodos para outros objetos, facilitando na modelagem e reutilização dos componentes.

```
class Hello extends React.Component {  
  render(){  
    return (  
      <h1>Hello World!</h1>  
    )  
  }  
}  
  
ReactDOM.render(  
  <Hello />,  
  document.getElementById("content")  
)
```

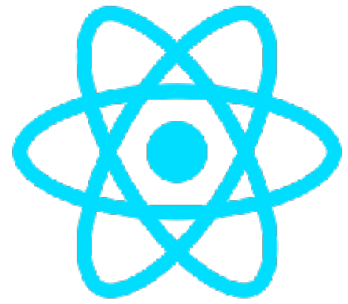
Nós iremos adotar essa abordagem em nossos exemplos.



Entendo o React - Usando Javascript e HTML

Abaixo definimos um dicionário de dados que vai chamar o Javascript dentro do método Render.

```
var carros = ['astra', 'civic', 'fusion'];
ReactDOM.render(
  <div>
    {
      carros.map(function (carro) {
        return <li>{carro}</li>
      })
    }
  </div>,
  document.getElementById('carros')
);
```

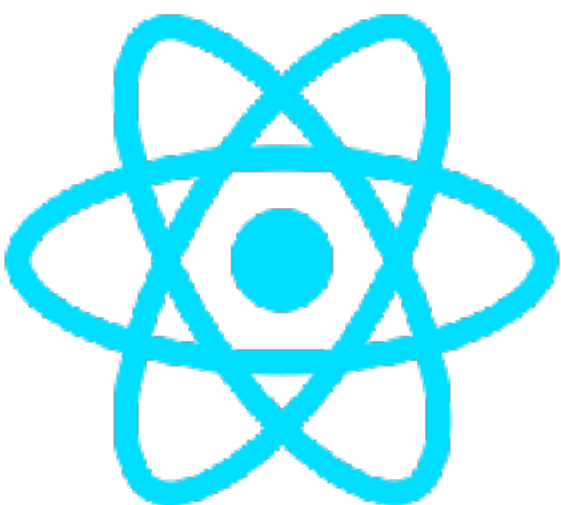


Entendo o React - Props

Quando utilizamos nossos componentes no React, nós podemos adicionar atributos a eles, dos quais chamamos de **props**. Esses atributos ficam disponíveis para o nosso componente através do **this.props** e podem ser usados no método **render**.

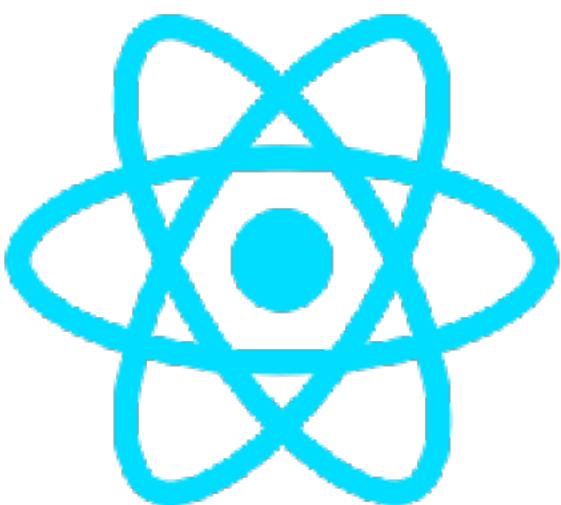
```
class Hello extends React.Component{
  render(){
    return (
      <h1> Hello {this.props.nome} !</h1>
    );
  }
}
ReactDOM.render(<Hello nome='alexandre' />,
  document.getElementById('root')
);
```

Se observamos o exemplo, vamos ver que o nosso código recebe uma variável do tipo nome.



Exemplo simples de um código em React

```
1 <html>
2   <head>
3     <meta charset="UTF-8" />
4     <title>Hello React!</title>
5     <script src="build/react.js"></script>
6     <script src="build/react-dom.js"></script>
7     <script src="https://cdnjs.cloudflare.com/ajax/libs/babel-core/5.8.23/browser.min.js"></script>
8   </head>
9   <body>
10    <div id="example"></div>
11    <script type="text/babel">
12      ReactDOM.render(
13        <h1>Hello, world!</h1>,
14        document.getElementById('example')
15      );
16    </script>
17  </body>
18 </html>
```

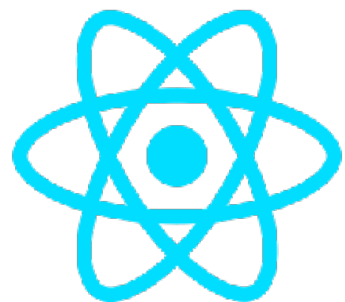


Criando o nosso primeiro exemplo simples

- 1. Baixe o React Starter Kit da seguinte URL: <https://react-cn.github.io/react/downloads.html>**
- 2. Descompacte o arquivo na pasta que deseja iniciar o seu projeto;**
- 3. Este arquivo contém as bibliotecas básicas do React e também muitos exemplos.**

A sua pasta deverá conter esses arquivos

Name	
▶	build
▶	examples
	README.md

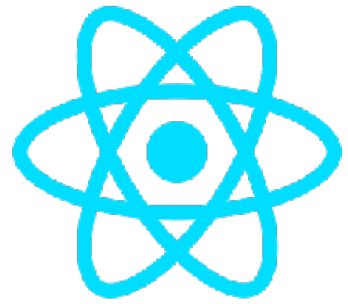


Criando o nosso primeiro exemplo simples

alexandre.rosa@unigranrio.edu.br alexrosa@gmail.com

1. Reproduza o código abaixo no seu computador

```
1 <html>
2   <head>
3     <meta charset="UTF-8" />
4     <title>Hello React!</title>
5     <script src="build/react.js"></script>
6     <script src="build/react-dom.js"></script>
7     <script src="https://cdnjs.cloudflare.com/ajax/libs/babel-core/5.8.23/browser.min.js"></script>
8   </head>
9   <body>
10    <div id="example"></div>
11    <script type="text/babel">
12      ReactDOM.render(
13        <h1>Hello, world!</h1>,
14        document.getElementById('example')
15      );
16    </script>
17  </body>
18 </html>
```

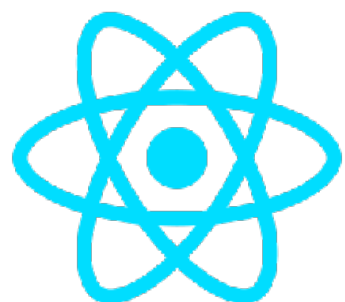


Criando o nosso primeiro exemplo simples

1. Declaração das bibliotecas

```
<head>  
  <script src="build/react.js"></script>  
  <script src="build/react-dom.js"></script>  
  <script src="https://cdnjs.cloudflare.com/ajax/libs/  
babel-core/5.8.23/browser.min.js"></script>  
</head>
```

2. A lib babel-core é usada para compilar o nosso código JavaScript com modo de compatibilidade para as versões (ES2015 e ES2016);



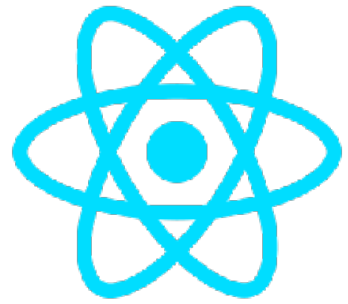
Criando o nosso primeiro exemplo simples

1. Dentro da tag `<body>` podemos ver como ficará o nosso código em React.

```
9  <body>
10    <div id="example"></div>
11    <script type="text/babel">
12      ReactDOM.render(
13        <h1>Hello, world!</h1>,
14        document.getElementById('example')
15      );
16    </script>
17  </body>
```

2. Na tag script deixamos claro que iremos usar abordagem (text/babel).

3. O método ReactDOM.render(...) será o responsável por renderizar o nosso componente no navegador. Repare que ele será renderizando na `<div id="example">`.



Criando o nosso primeiro exemplo simples

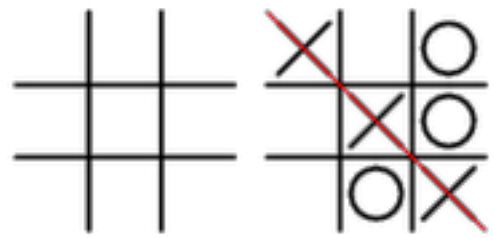
- 1. Agora crie o seu próprio Hello World;**
- 2. Vamos aumentar o desafio, faça com que o React imprima - Hello World. Hoje é (data de hoje);**

Parte 2

Desenvolvendo um game.



**Criando um jogo da Velha
(tic tac toe)**



Vamos começar...

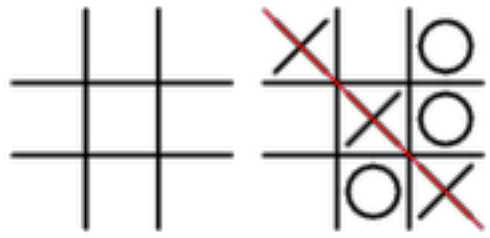
1. Primeiro crie o seu arquivo .html;

2. Inclua as referências do React no arquivo, usando a tag `<script></script>`, conforme abaixo:

```
<script src="react/react.js"></script>  
<script src="react/react-dom.js"></script>  
<script src="https://cdnjs.cloudflare.com/ajax/libs/babel-core/5.8.23/  
browser.min.js"></script>
```

3. Crie uma div com o id = “root”, conforme exemplo:

```
<div id="root"></div>
```

Vamos começar...

1. Primeiro crie o seu arquivo .html;

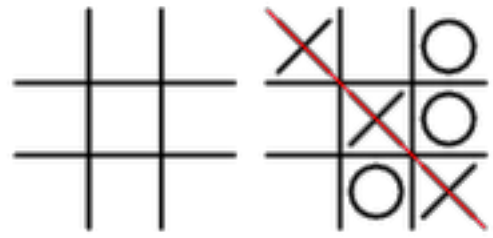
2. Inclua as referências do React no arquivo, usando a tag `<script></script>`, conforme abaixo:

```
<script src="react/react.js"></script>
<script src="react/react-dom.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/babel-core/5.8.23/
browser.min.js"></script>
```

3. Crie uma div com o id = “root”, conforme exemplo:

```
<div id="root"></div>
```

**4. Após crie uma tag `<script type="text/babel">` e insira o seu código dentro desta tag.
Lembre-se de fechá-la no final usando `</script>`**



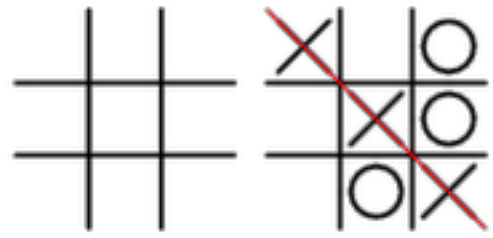
Criando o tabuleiro

1. Vamos criar um classe que irá representar o nosso tabuleiro, que será composto por 3 componentes:

1. Square (quadrados); - responsável pela renderização e ação do jogador (botões);

2. Board (tabuleiro) - responsável por renderizar a área do jogo (squares);

3. Game (Jogo) - responsável por renderizar o tabuleiro e gerenciar os elementos do jogo.

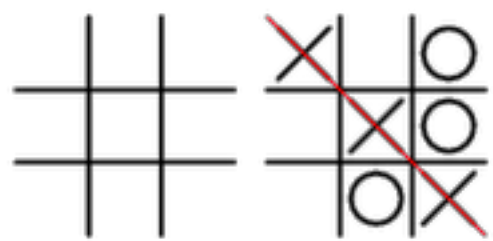


Criando o tabuleiro

1. Vamos criar a classe Square;
2. Agora vamos criar a Classe Board;
3. Vamos criar a classe Game que deverá instanciar os nosso tabuleiro.

Próximo Jogador: X

0	1	2
3	4	5
6	7	8



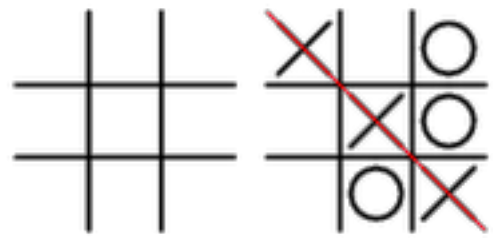
Criando o tabuleiro - classe Square

```
class Square extends React.Component{
  render(){
    return(
      <button className="square" onClick={() => alert('click')}>
        {this.props.value}
      </button>
    );
  }
}
```

Repare que nessa classe estamos criando uma arrow function (recurso do ES6)

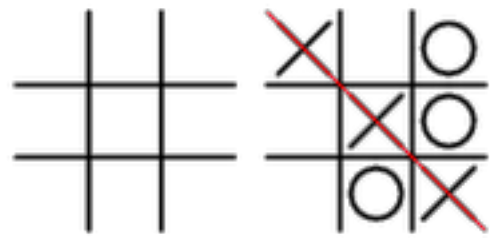
Próximo Jogador: X

0	1	2
3	4	5
6	7	8



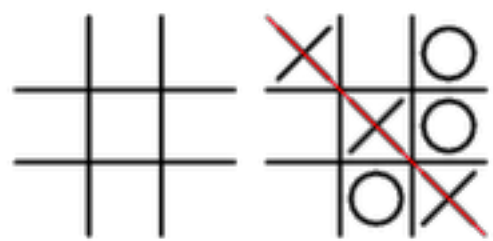
Criando o tabuleiro - classe Board

```
class Board extends React.Component{
  renderSquare(i){
    return <Square value={i} />;
  }
  render(){
    const status = 'Próximo Jogador: X';
    return(
      <div>
        <div className="status">{status}</div>
        <div className="board-row">
          {this.renderSquare(0)}
          {this.renderSquare(1)}
          {this.renderSquare(2)}
        </div>
        <div className="board-row">
          <!-- repita o mesmo passo da coluna anterior, incrementando os
números -->
        </div>
        <div className="board-row">
          <!-- repita o mesmo passo da coluna anterior, incrementando os
números -->
        </div>
      </div>
    );
  }
}
```



Criando o tabuleiro - classe Game

```
class Game extends React.Component{
  render(){
    return (
      <div className="game">
        <div className="game-board">
          <Board />
        </div>
        <div className="game-info">
          <div>{/* status */}</div>
          <ol>{/* TODO */}</ol>
        </div>
      </div>
    );
  }
}
```

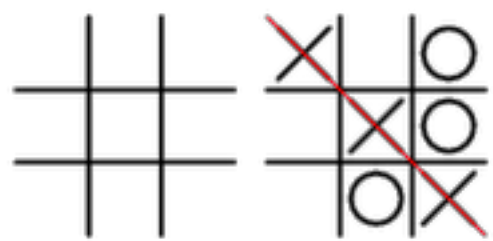


Criando o tabuleiro - Adicionando dinâmica ao jogo

Precisamos pegar o estado do componente, para armazenar o estado do botão clicado (selecionado). Para isso precisamos usar um recurso do React chamado de **this.state**.

1. Para isso vamos precisar alterar a nossa classe Square e adicionar um construtor.

```
class Square extends React.Component{  
  constructor(props){  
    super(props);  
    this.state = {  
      value: null,  
    };  
  }  
}
```

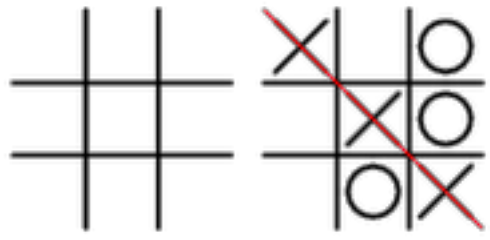



Criando o tabuleiro - Adicionando dinâmica ao jogo

2. Após vamos precisar alterar a função clique para armazenarmos o estado do botão clicado. Veja o exemplo abaixo:

```
class Square extends React.Component{
  constructor(props){
    super(props);
    this.state = {
      value: null,
    };
  }
  render(){
    return (
      <button className="square" onClick= {() => this.setState({value: 'X'})}>
        {this.state.value}
      </button>
    );
  }
}
```

Sempre que o evento `this.setState` for chamado, uma atualização do componente será realizada, fazendo com que o React atualize o valor do componente renderizado.

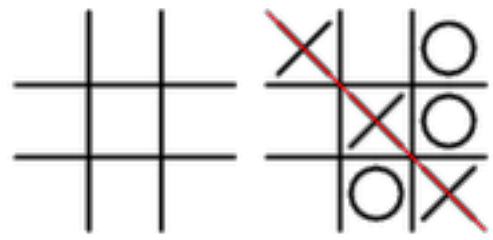


Melhorando o controle do Jogo - Board

Agora temos a construção básica do nosso jogo funcionando. Porém, o estado está encapsulado em cada quadrado (Square). Para realmente fazermos um jogo real, vamos precisar verificar se o jogador ganhou o jogo e alternar o X e O em cada área selecionada (square).

1. Vamos alterar o nosso construtor da classe Board

```
constructor(props) {  
  super(props);  
  this.state = {  
    squares: Array(9).fill(null),  
  };  
}
```



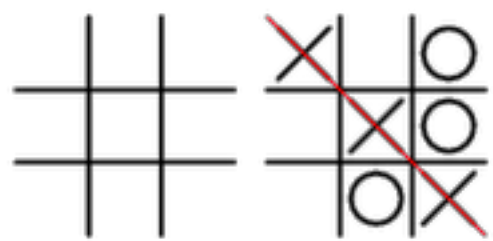
Melhorando o controle do Jogo - Board

2. Vamos alterar o método renderSquare(i) que antes era assim:

```
renderSquare(i){  
  return <Square value={i} />;  
}
```

3. E agora ficará assim:

```
renderSquare(i){  
  return <Square value={this.state.squares[i]} />;  
}
```

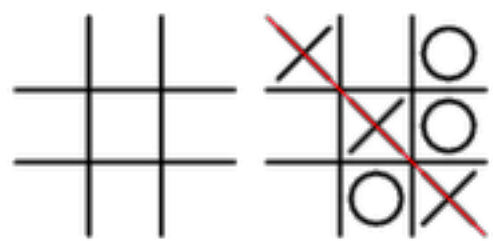


Melhorando o controle do Jogo - Board

4. Agora precisamos mudar o comportamento de quando o componente é clicado

```
renderSquare(i){  
  return (<Square  
    value={this.state.squares[i]}  
    onClick={() => this.handleClick(i)}  
  />  
);  
}
```

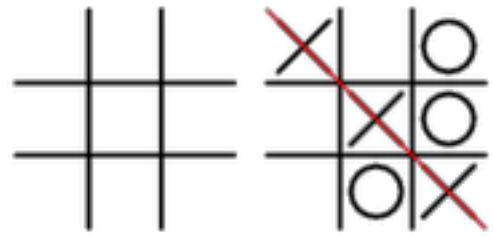
Repare que agora estamos passando 2 props da classe Board para a classe Square: **value** e **onClick**



Melhorando o controle do Jogo - Board

5. Implementando método handleClick();

```
//seta o valor de um square com X
handleClick(i){
  //foi usado o método slice para copiar o array
  const squares = this.state.squares.slice();
  squares[i] = 'X';
  this.setState({squares: squares});
}
```

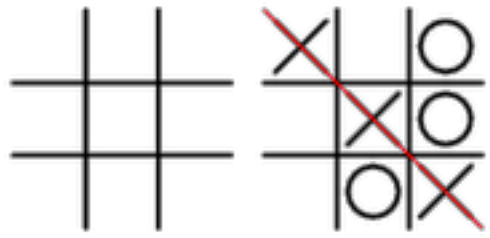


Criando componentes funcionais

React é uma linguagem tão dinâmica que suporta componentes como declarações de funções

```
function Square(props) {  
  return (  
    <button className="square" onClick={props.onClick}>  
      {props.value}  
    </button>  
  );  
}
```

Como agora o objeto Square se tornou uma função, temos que tomar cuidado para não usar mais o **props** fazendo referência a **this**.

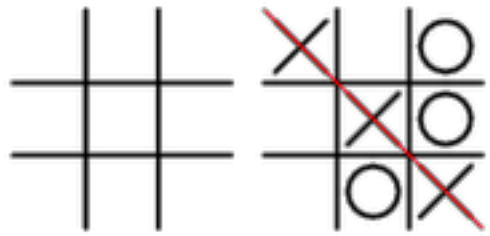


Corrigindo os defeitos

Até agora apenas o jogador X está jogando. Vamos alterar o nosso código para suportar também o jogador O.

Como transformamos a classe Square em função, vamos alterar a classe Board para incluir nela o construtor. Conforme exemplo abaixo:

```
class Board extends React.Component{
  constructor(props){
    super(props);
    //criar a propriedade square
    this.state = {
      squares: Array(9).fill(null),
      xIsNext: true;
    };
  }
}
```

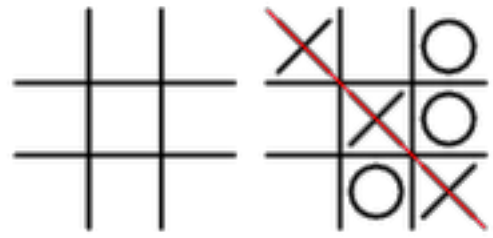



Corrigindo os defeitos

Agora, vamos alterar o método handleClick para controlarmos as jogadas.

```
handleClick(i){  
  //foi usado o método slice para copiar o array  
  const squares = this.state.squares.slice();  
  squares[i] = this.state.xIsNext ? 'X' : 'O';  
  this.setState({  
    squares: squares,  
    xIsNext: !this.state.xIsNext;  
  });  
}
```

Repare que agora estamos controlando quem está jogando se é X ou O.

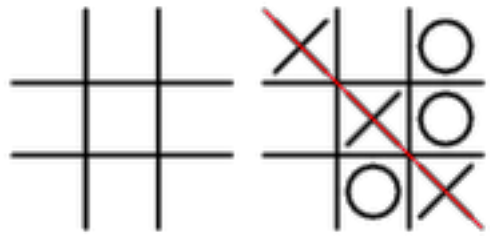


Corrigindo os defeitos

Vamos atualizar o status da jogador, informando quem será o próximo a jogar.

Para isso, vamos alterar o método `render()` da classe `Board`. Conforme exemplo abaixo:

```
render(){  
  const status = 'Próximo Jogador: ' + (this.state.xIsNext ? 'X' : 'O');
```

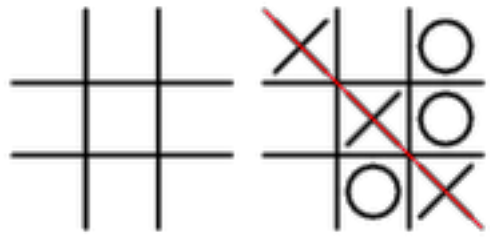


Declarando um vencedor

Adicione essa função helper que irá calcular o vencedor do jogo. Essa função poderá ser incluída no início ou final do seu script, conforme a imagem abaixo:

```
<script type="text/babel">
  //função para calcular o vencedor do jogo
  function calculaVencedor(squares){
    const linhas = [
      [0,1,2],
      [3,4,5],
      [6,7,8],
      [1,4,7],
      [2,5,8],
      [0,4,8],
      [2,4,6]
    ];

    for (let i=0; i < linhas.length; i++){
      const [a,b,c] = linhas[i];
      console.log(' valor da cel: '+linhas[i] );
      if (squares[a] && squares[a] === squares[b] && squares[a] ===
squares[c]){
        return squares[a];
      }
    }
    return null;
  }
  // DEMAIS CLASSES VIRÃO ABAIXO.
```

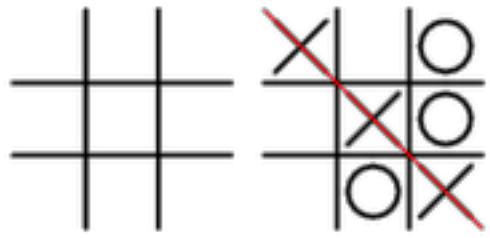


Declarando um vencedor

Agora precisamos alterar o status do nosso jogo e também bloquear o próximo jogador, caso tenhamos um vencedor. Então, vamos lá!

1. Na classe Board altere a função render() incluindo esse código no lugar do anterior: const status = 'Próximo jogador...'.

```
render(){
  const vencedor = calculaVencedor(this.state.squares);
  let status;
  if (vencedor){
    console.log('encontrou o vencedor');
    status = 'Vencedor: ' + vencedor;
  } else {
    status = 'Próximo jogador: ' + (this.state.xIsNext ? 'X' : 'O');
  }
  // abaixo deverá ficar a implementação do método return
```



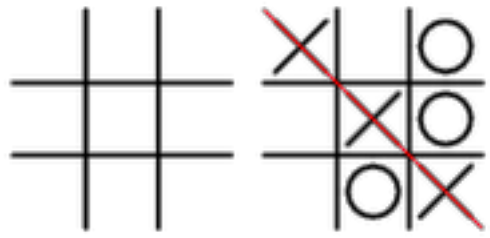
Declarando um vencedor

2. Agora precisamos alterar a função handleClick(i), para evitar que um próximo jogador jogue, caso tenhamos um vencedor.

A solução será fácil, basta inserir um controle (if) fazendo uma chamada a função calculaVencedor(squares) passando o nosso array como parâmetro. Veja o exemplo abaixo:

```
//caso tenha um vencedor a função abaixo irá impedir do próximo jogador jogar.  
if (calculaVencedor(squares) || squares[i]){  
    return;  
}
```

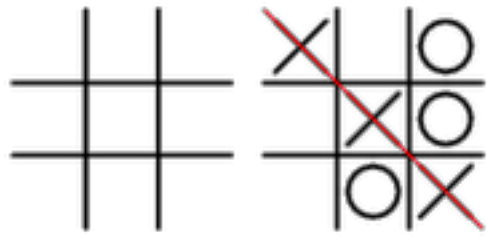
OBS: O comando return fará com que os demais comandos abaixo não sejam executados. Agora observe o resultado final do nosso método.



Declarando um vencedor

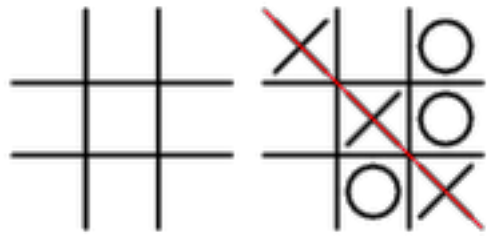
3. Como ficou o nosso método handleClick(i) após inserirmos um controle condicional (if).

```
//seta o valor de um square com X
handleClick(i){
  //foi usado o método slice para copiar o array
  const squares = this.state.squares.slice();
  //caso tenha um vencedor a função abaixo irá impedir do próximo jogador jogar.
  if (calculaVencedor(squares) || squares[i]){
    return;
  }
  squares[i] = this.state.xIsNext ? 'X' : 'O';
  this.setState({
    squares: squares,
    xIsNext: !this.state.xIsNext,
  });
}
```



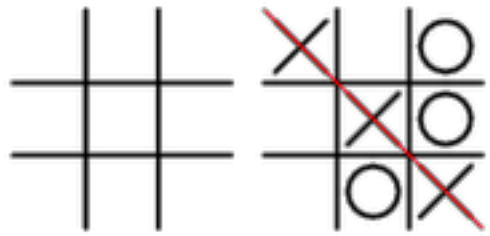
Parabéns!!!

Você conseguiu produzir o seu primeiro jogo com React.



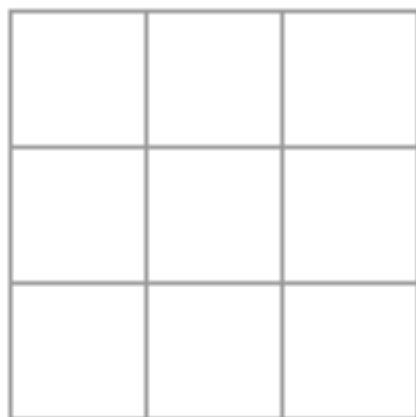
Parte 3

Vamos aprimorar o nosso game desenvolvendo alguns controles adicionais.



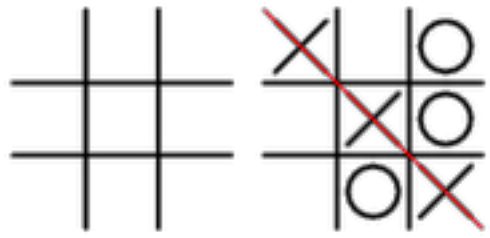
Criando novos controles

Você deve ter percebido que o nosso jogo já consegue definir um vencedor, porém não temos um controle (botão) para reiniciar a partida ou mesmo limpar o nosso tabuleiro. Para fazermos isto, precisamos fazer um refresh na página, ou seja reiniciar o estado da tela.



Próximo jogador: X

Jogar

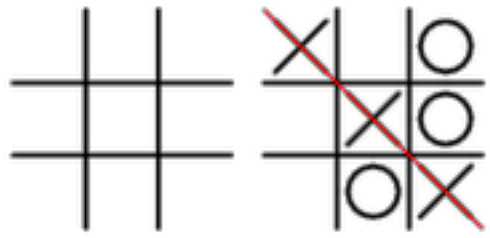


Criando novos controles

1. Primeiro vamos re-organizar o nosso jogo. Vamos transferir algumas responsabilidades que deveriam estar na classe **Game** e que estão na classe (componente) **Board**.
2. Sendo assim, primeiro passo a ser feito remover o construtor da classe **Board** e colocá-lo na classe **Game**.

Agora o escopo da nossa classe deverá ficar desse jeito

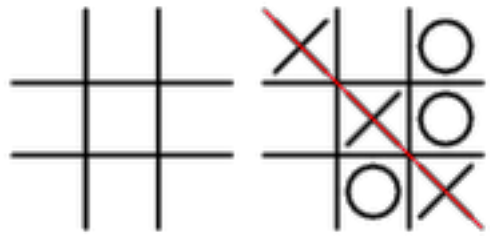
```
class Game extends React.Component {  
  constructor(props) {  
    super(props);  
    //criar a propriedade square  
    this.state = {  
      squares: Array(9).fill(null),  
      xIsNext: true,  
    };  
  }  
}
```



Criando novos controles

3. Agora vamos copiar o método handleClick(i) da classe **Board** para a classe **Game**, conforme o exemplo abaixo.

```
class Game extends React.Component {  
  constructor(props) {  
    super(props);  
    //criar a propriedade square  
    this.state = {  
      squares: Array(9).fill(null),  
      xIsNext: true,  
    };  
  }  
  
  //seta o valor de um square com X  
  handleClick(i) {  
    //foi usado o método slice para copiar o array  
    const squares = this.state.squares.slice();  
    //caso tenha um vencedor a função abaixo irá impedir do próximo jogador jogar.  
    if (calculaVencedor(squares) || squares[i]) {  
      return;  
    }  
    squares[i] = this.state.xIsNext ? 'X' : 'O';  
    this.setState({  
      squares: squares,  
      xIsNext: !this.state.xIsNext,  
    });  
  }  
}
```

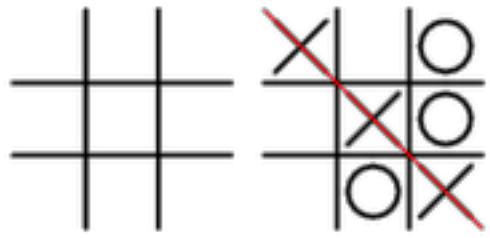


Criando novos controles

4. Agora vamos copiar o controle do status que estava sendo feito no método `render()` na nossa classe **Board** para a classe **Game**. Siga o exemplo abaixo.

```
class Game extends React.Component {  
  render() {  
    const current = this.state.squares;  
    const vencedor = calculaVencedor(this.state.squares);  
    let status;  
  
    if (vencedor) {  
      console.log('encontrou o vencedor');  
      status = 'Vencedor: ' + vencedor;  
    } else {  
      status = 'Próximo jogador: ' + (this.state.xIsNext ? 'X' : 'O');  
    }  
    //abaixo deverá conter a chamada a return()  
  }  
}
```

Copie o código que estava inserido no método `Board.render()` para a classe `Game.render()`, conforme o exemplo abaixo. Observe que agora o método `render()` da classe `Game` ficou parecido com o da classe `Board`.



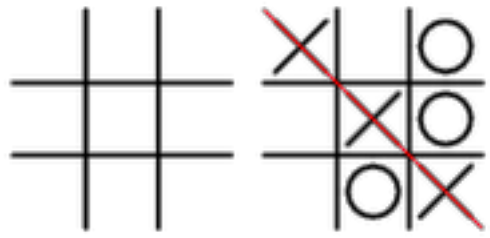
Criando novos controles

5. Agora a nossa classe Board deverá ficar apenas com dois métodos, são eles:

- renderSquare(i);
- render();

Visão geral da classe Board

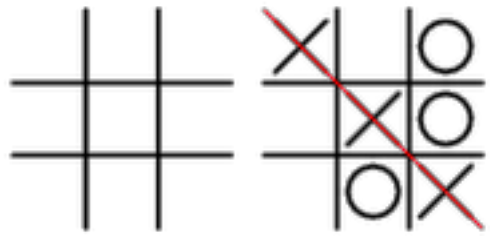
```
class Board extends React.Component {
  renderSquare(i) {
    return (
      <Square
        value={this.props.squares[i]}
        onClick={() => this.props.onClick(i)} />
    );
  }
  render() {
    return (
      <div>
        <div className="board-row">
          {this.renderSquare(0)}
          {this.renderSquare(1)}
          {this.renderSquare(2)}
        </div>
        <div className="board-row">
          {this.renderSquare(3)}
          {this.renderSquare(4)}
          {this.renderSquare(5)}
        </div>
        <div className="board-row">
          {this.renderSquare(6)}
          {this.renderSquare(7)}
          {this.renderSquare(8)}
        </div>
      </div>
    );
  }
}
```



Criando novos controles

5.1 Ainda na classe **Board** precisamos ainda fazer algumas alterações sutis, porém importantes.

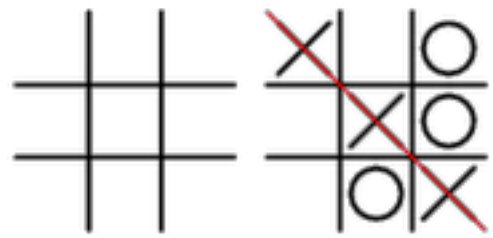
- No método `renderSquare` precisamos alterar a renderização de **Square**, mudando a chamada anterior que estava `value={this.state.squares[i]}` para `value={this.props.squares[i]}`
- No método `render()` precisaremos remover a declaração e controle do status e mover ele para a classe `Game`, que passará agora ser responsável por controlar o status do jogo.
- Ainda no método `render()`, nós também iremos remover a tag `<div className="status">{status}</div>`. Como dissemos, esse controle passará a ser feito pela classe `Game`.



Criando novos controles

5.2 Agora veja como ficou a nossa classe Board.

```
class Board extends React.Component{
  renderSquare(i){
    return (<Square
      value={this.props.squares[i]}
      onClick={() => this.props.onClick(i)} />
    );
  }
  render(){
    return(
      <div>
        <div className="board-row">
          {this.renderSquare(0)}
          {this.renderSquare(1)}
          {this.renderSquare(2)}
        </div>
        <div className="board-row">
          {this.renderSquare(3)}
          {this.renderSquare(4)}
          {this.renderSquare(5)}
        </div>
        <div className="board-row">
          {this.renderSquare(6)}
          {this.renderSquare(7)}
          {this.renderSquare(8)}
        </div>
      </div>
    );
  }
}
```



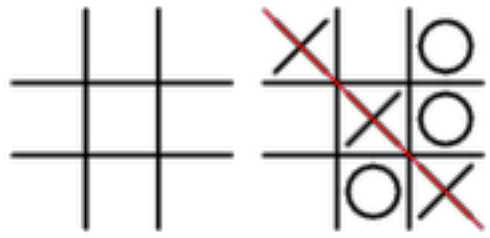
Criando novos controles

6. Classe **Game**, agora faremos os ajustes necessários.

1. Vamos criar um novo método chamado `reset()` dentro da classe `Game`. Siga o exemplo abaixo:

```
//limpa os dados da janela.  
reset() {  
  this.setState({  
    squares: Array(9).fill(null),  
    xIsNext: true,  
  })  
  console.log('resetando a tela');  
}
```

Vamos explicar: O React trabalha com o controle de estado dos seus componentes, logo para limparmos a nossa tela, precisamos limpar o nosso array ou seja, resetar os valores informados pelo usuário. Para isso, foi criada uma nova variável e usado o comando: `this.setState()`. Esse comando irá resetar o estado do nosso componente `Game`.



Criando novos controles

6. Classe **Game**, agora faremos os ajustes necessários.

2. Vamos alterar o método **render()** da classe **Game**, pois agora precisamos mudar a chamada do componente **<Board>**. Siga o exemplo apresentado.

```
return(  
  <div className="game">  
    <div className="game-board">  
      <Board  
        squares={current}  
        onClick={(i) => this.handleClick(i)}  
      />  
    </div>  
    <div className="game-info">  
      <div>{status} </div>  
      <div><button onClick={() => this.reset()}>Jogar</button></div>  
    </div>  
  </div>  
)
```

Observe que agora precisamos passar a nossa lista (squares) e também o handle da chamada onClick() para o componente (classe) Board.

Incluimos também um botão e uma chamada para a nossa função reset()

Referências

Github:

- <https://github.com/alexrosa> - Aqui você poderá fazer o download dos exemplos.

O material utilizado nesta aula foi extraído do site oficial do ReactJS. Lá você poderá encontrar bastante conteúdo sobre, inclusive o tutorial do jogo da velha (tic tac toe).

Sites:

- Website oficial - <https://reactjs.org/>;
- Tutorial oficial - <https://reactjs.org/tutorial/tutorial.html>

React Native:

- <http://facebook.github.io/react-native/>

NPM (pacote que facilita a criação das apps):

- <https://www.npmjs.com/package/create-react-app>

Obrigado

Alexandre Rosa
alexrosa@gmail.com