Tic-Tac-Toe Step 3: Adding useState to the game and setting states for the squares

# Introduction:

In this exercise we will introduce useState to our game. useState will be very important part of the game and how it works. We will create an useState array, which we will use in the future to make the moves of the game, we will link the array entries to each of the Board components boxes (9 rendered Square components).

# Adding useState:

We will start in our base component. First thing that we’re going to do, is import useState and create an array called initialBoard. We will place that at the top of the page, under the imports:

import React from "react";

import './TicTacToe.css';

import { Board } from "./Board";

import { useState } from "react";

const intialBoard=['', '', '', '', '', '', '', '', '',];

export const TicTacToe = () => {

    return (

        <div>

            <div className="game">

                <h1>Tic-Tac-Toe</h1>

                <Board />

            </div>

        </div>

    );

}

This array will be our “**database**”, each empty entry being a box on the gameboard (**3 x 3**). Here we will make our moves and alter the data while the game is in progress. Next we will add a state inside our base component called gameState. We will set the **initialBoard** array as the states **initialState**:

const [gameState, setGameState] = useState(intialBoard);

Alternatively, we could set the empty array as the initialState, but for the sake of the future we will **store the empty array on its own**. The reason for that is, in the future we will need to clear the array in multiple parts of project (**setting the board to its initialState**).

# Assigning the gameState array to the Board:

As you probably know, in arrays the **item ID’s start at a 0 instead of 1**. That is critical information because we will need to alter the slots by their **ID**. Below is an example of the ID’s of our initialBoard and how they will be arranged on the game:

[

'0', '1', '2',

'3', '4', '5',

'6', '7', '8',

];

Where we have rendered the board in the base component, pass the gameState to our board component using props:

<Board gameState={gameState}/>

Now we’ll move to our Board component. Remember that in the previous exercise we created the base for our board using the Square components. But now the squares are just hard coded X’s that we added as the placeholders:

return (

    <div className={classes}>

        <h1>X</h1>

    </div>

)

So, we’ll add the gameState to our board and assign an entry from its array **to each box** in the right order.

Pass the gameState to the Board component as a prop. We'll use squirrely brackets to get the gameState from the props right away, so we don't need to always write “props” Infront of the state.

export const Board = ({ gameState }) => {

return (

Now that we can use the state in the Board component, for every Square component that we have rendered add the gameState. From **top to bottom**, from **0 to 8**, like shown below:

export const Board = ({ gameState }) => {

    return (

      <>

        <div className="row">

          <Square

            className="b-bottom-right"

---------------------

            state={gameState[0]}

---------------------

          />

          <Square

            className="b-bottom-right"

---------------------

            state={gameState[1]}

---------------------

          />

          <Square

\*\*\*ETC\*\*\*

I recommend for the sake of readability, **add the state on its own line**, under the className.

Now we can **alter individual boxes of the board** (making the moves in the game), instead of every slot being a **hard coded component**.

# isXTurn:

Lastly, we will learn a bit of how we will code the switch of turns when the game is playable eventually. Again, we’re going to do that with useState, but with a **boolean** value this time. In the next exercise, we will make the function that determines the current player and **makes the switch between turns**. But we will set up the state now so it will be ready to go.

Add a state under our gameState, named **isXTurn**. Initial state **true**, so first turn will go to **X**, since by the official Tic-Tac-Toe rules the first turn is made by X. That will make O’s turn to happen when the isXTurn state is set to **false.**

const [isXTurn, setIsXturn] = useState(true);

We will continue to start the **onSquareClick** function in the next exercise. That will be on charge of switching the state that determines player turns we just made. The function will also be used on making the moves on the board (and a couple of more functions that will be introduced later). **See you there!** 😊