Tic-Tac-Toe Step 4: onSquareClick

# Introduction:

In this exercise we’re going to create maybe the **most important** function of the game, **onSquareClick**. The function will **be in charge** of the course of the game, will it continue and update the changes on the board? Determine whose turn it is? Stop the game if there is a winner, and more. We will not be doing all the features in this one exercise, but we will update it in the later steps. We will assign this function to our squares with onClick, as the name hints, it will execute when the **squares are clicked**.

# onSquareClick:

Start by creating an **arrow function** in the base component, **above the return tag and below the states**:

export const TicTacToe = () => {

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

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

    const onSquareClick = () => {

    }

    return (

        <div>

            <div className="game">

                <h1>Tic-Tac-Toe</h1>

                <Board gameState={gameState}/>

            </div>

        </div>

    );

}

Next, we will create a copy of the gameState array, named **strings**, so we can use that to set the state after the turns. We will do that with the JavaScrip method, **Array.from()**:

function onSquareClick = () => {

    let strings = Array.from(gameState);

}

# if statement:

We will add an **if statement** under the copy. With this statement we will check if the square we try to make the move in, is empty. If **not**, the function will stop. If the square is not empty, that means a player has made their move on that spot (occupied with X or O). **With this statement, we can prevent the overwriting of moves**.

We will check the specific square with the **item index** that we’re already familiar with:

if (strings[index] !== ''){

    return;

}

As you remember, the item index is the automatic index of entries of an array, our indexes being **from 0 to 8**.

# Switching turns:

In the last exercise, we created the **isXTurn** state that checked the turn of the player (**true=X false=O**). We will now use that inside our onSquareClick function, to change the turn of the player after a move is made. We will do that with the already familiar JavaScript **ternarary operator.**

We will check if the isXTurn state is set to **true**. If it is, the function will insert an **X** on the square the player clicks, **otherwise** it will insert an **O**. Again, we will use the item index of the gameState copy to know the specific square being clicked:

if (strings[index] !== ''){

    return;

}

strings [index] = isXTurn ? 'X' : 'O';

Under the ternary operator we will set the gameState and isXTurn. Setting the gameState to the strings copy and the isXTurn to **false**:

strings [index] = isXTurn ? 'X' : 'O';

setGameState(strings);

setIsXTurn (!isXTurn);

# Assigning the function:

Before we move on from the function, we have to give an argument to it. To add this function to our individual squares and **make them clickable**, we must add the **item index** of the array inside the brackets:

function onSquareClick = (index) => {

We also have to add this in the Board component tags where we render it, the same way we did with the state:

return (

    <div>

        <div className="game">

            <h1>Tic-Tac-Toe</h1>

            <Board gameState={gameState} onSquareClick={onSquareClick}/>

        </div>

    </div>

);

We also have to pass the function to our Board component:

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

At the moment, our rendered Square components inside the Board component look like this:

<Square

className="b-bottom-r"

  state={gameState[0]}

/>

Just like we did in the previous step when we gave each of the squares a unique state, we will pass them the function we just made, and call the function when the square is clicked with the square's index. Add the function to all the squares with the **same index the state has**, from 0 to 8:

<Square

  className="b-bottom-right"

  state={gameState[0]}

  onClick={() => onSquareClick(0)}

/>

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

# onClick:

Now what we have to do, is move to our Square component and make it clickable and display the turns instead of the hard coded X’s. Since we’re getting the function with props, we will simply use **props.onClick**.

Just like we would add onClick to a button:

<button onClick={function}></button>

We will add it to our Square components div element exactly the same way:

<div className={classes} onClick={props.onClick}>

We have to do one last thing to before we finally see some functionality in our browser. Again, we’re using props so, with the gameState that we have already assigned to the individual squares in the Board component:

<Square

  className="b-bottom-right"

  state={gameState[0]}

  onClick={() => onSquareClick(0)}

/>

We will **replace** the header with **props.state**:

return (

    <div className={classes} onClick={props.onClick}>

        {props.state}

    </div>

)

# Browser:

Move to your browser:

A picture containing text, shoji

Description automatically generated

We’re **finally** at the point where we can write on the board! You should see that after the first move made by **X** the ternary operator will change it to **O**, and so on. Even though the game is not functional yet and lacks the game logic (like winner announcement etc.), we can continue to develop the game and **see the progress visually**, which is far more rewarding than just writing code and not have anything to show for it.

In the next step we can finally start adding the game logic to the project and add the winner calculation. **See you there!** 😊