Tic-Tac-Toe Step 6: Conditional rendering

# Introduction:

In this exercise we will be adding **conditional rendering** using our status state. As you remember from our last exercise, we just render the status under our board. Now we’re going to change that, so every option **has its own render**. We’re also going to add different **buttons** to clear the board and add some coloring to the status depending on the outcome (green winner announcement, black draw announcement etc).

# Conditional rendering:

Conditional rendering means that we will give a condition, if that condition is **true**, a render will happen, if its **false**, nothing happens.

The condition will be **inside squirrely brackets**, followed by brackets where the code that will be executed is. This way we can edit, for example the **color** of specific conditions.

\*\*\*EXAMPLE\*\*\*

{ifThisIsTrue && (

    <h1 style={{color: "green"}}>We will render this</h1>

)}

Above is an example of how we’re going to render with a condition. We have a function, if it’s **true**, we’re going to render the header. If its **false**, nothing happens. To render it, **the condition must be true**. If we want to render something where we **want** the function to be false, we’re going to add an **exclamation mark** in front of the function:

{!ifThisIsTrue && (

So, because the **function above is false**, our **condition will be true**, and the render will happen.

# Conditional rendering with multiple elements:

When we want to render **two or more** HTML elements, we need to add <> and </> tags. The reason for that is, in React, you can **return only one root element**. In this case, the empty tags will work as our root element and **make it possible to render them both in the same condition** and spare us the trouble of making duplicate conditions.

Below is an example of the tags in use:

\*\*\*EXAMPLE\*\*\*

{sameFunction && (

    <span>Otherwise, we would have to add two of same conditions</span>

)}

{sameFunction && (

    <button>To render these two elements</span>

)}

{sameFunction && (

    <>

    <span>But with the <>, and </> tags</span>

    <button>We can render them both in the same condition</button>

    </>

)}

# Current turn, a draw, and a clear board button:

# Current turn and a draw:

Now we’ll start adding the conditions to our base component, we’re going to make our status be visible when a winner **is not** found. First, lets remove the status render under our Board element:

return (

    <div>

        <div className="game">

            <h1>Tic-Tac-Toe</h1>

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

            {status}

        </div>

    </div>

);

We will be using the .includes method in both of our conditions, like we did in the last exercise where we stopped the onSquareClick function.

If the status doesn’t contain the word “Winner”, we will render the status below. That means the game is in progress, or the game has ended in a draw. We want the color of both options **remain black**, so we can render them both here with the same condition, **without adding** a style tag:

{!status.includes("Winner") && (

    <span>{status}</span>

)}

A picture containing calendar

Description automatically generated

# Clear game button:

At the moment we have no other option to clear the board (winner or not), but to refresh the whole page. We will add a button to reset the board mid game or in case of a draw (status not containing the word “Winner”).

Because this will be the **second element** to render in this condition, we will need the tags to work as our **root element**:

{!status.includes("Winner") && (

    <>

        <span>{status}</span>

    </>

)}

The button will be a bit trickier, since it has some functions to execute (clearing the board and resetting the turn). Obviously, we’re going use our setGameState and setIsXTurn states. And since we’re executing the functions with the **click of the button**, its going to be with the familiar onClick event.

If you don’t remember how to assign onClick to elements, below is an example:

<button onClick={() => {

} }></button>

**Inside**, reset the board (gameStato to initialBoard) and make the game start with **X**’s turn (setIsXTurn to **true**), and name the button to Clear board.

setGameState(intialBoard);

setIsXturn(true);

Now we have a button to clear the game **mid game**, or in a case of a **draw** without having to reset the whole page:

A picture containing calendar

Description automatically generated Calendar

Description automatically generated

# Winner and play again button with styling:

# Winner:

Let’s add our winner announcement first. If the status includes the word Winner, we will render the status, but in a green color. We will add the coloring with the **style tag** below:

{status.includes("Winner") && (

    <span style={{color: "green"}} >{status}</span>

)}

Now the Winner should be displayed in a nice green color:

A picture containing shape

Description automatically generated

# Play again button:

Last thing we’ll do, is add a green colored Play again button, when winner **is** declared.

Again, since there are two elements to render, we need to add the tags:

{status.includes("Winner") && (

    <>

        <span style={{color: "green"}} >{status}</span>

    </>

)}

The functionality of the button will be **identical** to our Clear game button, so you can go ahead and copy it.

<button onClick={() => {

    setGameState(intialBoard);

    setIsXTurn(true);

} }></button>

Last thing is to add the styling and name the button to Play again and setting the **background color** to **light green**:

<button style={{background: "lightgreen"}} onClick={() => {

    setGameState(intialBoard);

    setIsXturn(true);

} }>Play again</button>

Calendar

Description automatically generated

Now we have added some diversity on the status depending on the outcome.

The game is now fully working with the buttons to reset the board. The only part left on this course is to add a **scoreboard** that keeps track of the games won by each player. The scoreboard will also contain a button to reset the scores. We will add them in the next, and **last** exercise. **See you there!** 😊