Tic-Tac-Toe Step 5: Game logic

# Introduction:

In this exercise we will add the **winner calculation** to our game. We will be copying the calculation itself since it’s a bit too complicated to include as a step in the exercise. Instead, we have added a file where the process is explained step by step. You can learn (**if interested**) about it [here](ds).

We will also add an useEffect hook that checks the status of the game with useState. Here, using the checkWinner function we will display the **current turn**, a **draw** or a **winner**.

Let’s start by copying the checkWinner function. Place the following code below our onSquareClick function:

const checkWinner = () => {

    const lines = [

        [0, 1, 2],

        [3, 4, 5],

        [6, 7, 8],

        [0, 3, 6],

        [1, 4, 7],

        [2, 5, 8],

        [0, 4, 8],

        [2, 4, 6],

    ];

    for (let i = 0; i < lines.length; i++) {

        const [a, b, c] = lines[i];

        if (gameState[a] && gameState[a] === gameState[b] && gameState[a]

=== gameState[c]) {

            return gameState[a];

        }

    }

    return null;

}

# useState & useEffect:

Now we will be adding a third state to our base component. It will simply be called status, initial state an empty string:

const [status, setStatus] = useState('');

This state will be **important and used in many places**. We will be using it to display the **turn of the current player**, a **winner** or a **draw**, and make the **game stop** to prevent making moves after winner is declared. Also adding buttons in the future that use this state to function.

Next, add a useEffect hook with **a dependency** that triggers when our gameState is updated (**meaning a move is made in the game**).:

useEffect(() => {

}, [gameState])

If you need a reminder how useEffect works you can check the Counters.jsx tutorial.

We will store the checkWinner function in a variable so we can **easily use it in our if statements**:

const winner = checkWinner();

Now, inside our useEffect hook, we’ll add an **if statement** that checks if a winner is found. We will do that by checking if the checkWinner function has returned **X** or **O** (winner found) or **null** (a draw).

If a winner **is** found, we will set our status state to display the winner. Because we want to set the state to include a string and a variable, we have to use **backticks** and a **dollar sign**:

if(winner){

setStatus(`Winner: ${winner}`)

}

To get our winner announcement visible, we will have to render it. So, under the Board element add the status:

<div>

    <div className=”game”>

        <h1>Tic-Tac-Toe</h1>

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

        {status}

    </div>

</div>

Now we should see the status pop up **if** a winner is found (before that, or in case of a draw we see nothing):

Calendar

Description automatically generated with medium confidence

# onSquareclick:

As you may have noticed, even when the winner is declared, you **can still make moves on the board**:

A screenshot of a cell phone

Description automatically generated with low confidence

To fix that we have to add a small if statement inside our onSquareClick function, **as the very first if statement**. We will use JavaScript **.includes** method to see if our status state contains the word **Winner** (meaning that our checkWinner function has found a winner for the game):

const onSquareClick = (index) =>{

    let strings = Array.from(gameState);

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

    if (status.includes("Winner")) {

        return

    }

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

    if (strings[index] !== '') {

        return;

    }

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

    setGameState(strings);

    setIsXChance(!isXChance);

}

Now the game “freezes” when a winner is found, and nothing is clickable.

# A draw and turn status:

Next, we’ll set the state to show if the game ends in a **draw**, and the **current turn of a player**. So, let’s continue our if statement with an **else if**:

if(winner){

    setStatus(`Winner: ${winner}`)

} else if () {

}

Remember that the **function will stop if a winner is found**, so what ever comes after the first if, will be executed only if there is no winner (In case of a draw, or a game in progress).

Inside the brackets, with the .includes method, we will check if our gameState array has any empty slots (**meaning the game is still in progress**), if it doesn’t the status will be set to display the draw announcement:

} else if (!gameState.includes('')){

    setStatus(`It's a draw`)

}

The last status we’ll add is the current turn. We will do that by adding else as the last condition. Because the function has continued here, that means that a winner is not found, but there is still empty slot/s in the gameState array (**game in progress**).

Because we have to **switch between** X’s turn and O’s turn, we will use the already familiar **ternary operator**. This state is a bit tricky since we’re using the ternary operator inside the status. Inside the setStatus brackets, **wrap everything in back ticks** (``) and add the ternary operator and string like shown below:

else {

    setStatus(`${isXTurn ? 'X' : 'O'}'s turn`)

}

A picture containing shoji

Description automatically generated

Now we should have the status state visible at all times.

We will end this part here, now our game is functional and playable, all that’s left to **add is buttons to clear the board** and a **scoreboard**. In the next exercise we will edit the status rendering a bit, add buttons and take a look at **conditional rendering**. **See you there!**😊