Tic Tac Toe approach

*// Object deconstructors*

const Box = (props) => {

const value = props.value *// is 'x'*

console.log(value) *// logs 'x'*

}

const Box = ({ value }) => {

console.log(value) *// logs 'x'*

}

1. Object deconstructors
   1. Seems quite useful; if we pass in an object as a parameter with just key names,
   2. And the input that was passed in is an object containing those keys,
   3. Then we will automatically generate variables that are assigned those property values of the passed-in object.

const [buttonVal, setButtonVal] = useState('-')

const changeButtonVal = () => {

let newButtonVal;

*if* (buttonVal === '-' || buttonVal === 'O') {

newButtonVal = "X"

} *else* newButtonVal = "O"

setButtonVal(newButtonVal)

}

1. Setter functions:
   1. Remember that state values are read-only.
   2. In order to pass in a new state, we need to first figure out what the new state is and pass that in wholecloth.
2. The approach:
   1. Top level parent components contain the setter and state information.
   2. The state information is initialized to an object
   3. And this object will be passed into the setter function whenever something needs to be updated.
3. Miri takes the board functionalities within the Board component
   1. Which includes things like:
      1. Checking for a winner
      2. Resetting the board
      3. Restarting the game
      4. Checking for player turn
      5. And useEffect that updates the status of the board (whether that’s resetting the board to a new board or checking the status of a winner).
4. Remember – in these cases where grids values and such are needed to be tracked
   1. An object or an array of values can be quite helpful!
   2. Don’t be afraid to store information.
5. Conditional rendering

*This is a conditional rendering operator. It only renders “p” with the message contained if gameOver is truthy.*

*return* (

{gameOver && <p>{msg}<p/>}

)