Tic-Tac-Toe Exercise With React

Video Transcript

Video 1 – Introduction To Tic-Tac-Toe Exercise

I'm sure you've all played Tic-Tac-Toe when you were a child. I know I did. So, in this exercise, we're going to simulate a Tic-Tac-Toe game. We're going to have two levels of web component. We're going to have the board, and then we're going to have the nine squares, which are going to be buttons, and they're going to be children of the board. Now, we need to keep track of state. We need to know whose turn it is, and that's going to be kept at board level. But then the child needs to know who's playing, who's going to make the next move. And so, we need to communicate between the parent and the child, the state. We'll need to use callbacks so that we pass a callback from the parent down to the child so the child can do what it needs to do, and then transfer the information back to the parent. So a lot of this is going to be synchronizing the nine children.

These are going to be the nine buttons and they need to keep track of their own state, whether they are an X or an O. And at the end of the game, you need to find out has somebody won or not. We need to check whether there is a line of Xs or a line of Os. So, there's quite a bit to this game. It's been fun making it. And it's actually quite good fun to play it as well. Although after a while you realize that you can always kind of win or you can at least make a draw out of it. You shouldn't lose. But it's a fun game. And it's great exercise in user interface design. There's a lot of management of where the buttons are placed. So, you'll be using a grid to manage those. And CSS plays a big role in making it look good. So, have fun with Tic-Tac-Toe. You can change the layout for yourself and make it look even prettier.

Video 2 – Tic-Tac-Toe Overview

So, we're going to build up a tic-tac-toe game like this. And we're going to build it from web components in React. And if you haven't played tic-tac-toe, The idea is we have this three-by-three grid and it's empty, to begin with. And there are two players. One has the X's and one has the O's. And we take it, in turn, to fill in the squares. And the idea is to get three in a line. So, you can either have them along a row, column, or diagonal. But they must be a kind of straight line. And it's a very simple game to actually play. But we're going to code it up. And that's going to teach us a lot about web components.

Video 3 – Generate Players On The Tic-Tac-Toe Board

So, let's start with a single web component. So, here's our first. It's just going to render this background here. That's going to be our game board if you'd like. So, let's take a look at the code that does that. We have two files. So, this first one is an HTML file that loads up some libraries. So, we're loading in the 'react.development.js', and the 'react-dom.development.js', and we're going to load in our file here, 'colorGame00.jsx'. So, let's see how that works. There's some other CSS that I've put in here, style sheets. So, we've got '@babel' up here. And also, we have some style sheets here. And the main one is this '.game-board'. And you'll see it's '600' by '600'. And the color, 'background-color:' was this kind of blue.

So, that's what you saw on the browser there. Let's take a look at the file that created that. So, we're going to create a web component called 'Board'. So, it's going to be launched into the DOM renderer. So, this is our web component. So, we have a definition up here in React. We start with a capital letter for the name of our component. And I'm doing it in the '=>' notation here. So, we have no arguments at the moment. And this is the body of our function, and we're just going to return some JSX. So, this is going to be React type code here where we're returning not even strings, but these are legal in JSX. And let's expand it, there's a little bit more code here. We're going to have a 'status' bar that we're going to fill in.

At the moment, we don't have anything there. So, that's our code. And this renders this 'Board' web component. So, that's basically it, at the moment. Now, let's put something else in here. So, for example, we'd like to keep track of which player is playing. So, we've got these Xs and Os. So, we have two players. So, let's put in, 'let player =". Let's just say, '1'. We'll have 0 and 1, 0 will be the Os, and player 1 would be the Xs, say. So, now we can type in something for, I'm giving the backticks. So, let's just say, ' `Player` ', and then let's put in the players. So, this will put a one in there, and it'll put it in the 'status'. Okay, so let's save that. And let's re-render our game board. Here we go. So, we've got Player 1 in there. Now, we'd like to have the player saved in this component.

We'd like to make it, make our component have state so that the game board if you'd like, will know which player is playing. Player 0 or Player 1? Let's do a little bit more editing, and just, let's take a look at what we want to do here. Well, this worked fine. But let's see what happens if we increase 'player'. For example, if we clicked on the 'Board'. So, how would we do that? Well, we'd need to put an onClick in this '< div >'. Let's do that. So, here, let's make some space in the '< div >'. And we'll put an 'onClick'. This is going to be code. So, now that's better. So, 'onClick'. And then it equals, we need another one of these to finish it, yeah. Now, that looks much better. So, this is 'onClick'. It needs to be a function. And that's what this is. And in fact, I can get rid of. Now, I need to be careful because yes, I'm in 'return' statement here says that's correct.

Let's save that. We're now in the 'onClick', we're going to increase, we're going to increment the 'player' by '1'. So, we're going to say, 'player = player + 1;'. That seems reasonable. And we'll put that into 'status' and see if it's rendered out. Okay, let's go and run that. So, let me reload. So, we've got Player 1, that's good. Let's bring up the div environment and we'll go into the sources. And I'm going to see outbreak in the 'onClick', okay? So, let's just make sure that so I'm going to

scoot across here just so you can see where we are. And let's click on this. So, the click certainly works. The 'player' now is '2'. And so, we should see 'player 2'. Let's let this run. No, didn't work. Why is that? Well, it's because React didn't re-render.

So, React keeps a shadow DOM. And if it sees that something that you're rendering has changed, then it should update, but it's not. And it's because we need to keep state in a special way in our component. So, what I'm saying is the 'Player' number, either 0 or 1. We need to keep that as state in this component. Now, there's a new way of doing that in functional components like this. And what we have to do is we need to use something called 'useState'. So, we say, 'const', and we'll have, we'll store the player's name. And we need to put the function that we're going to use to store it. And those, we're going to get from 'React.useState()'. And we can initialize the 'useState'. So, we do that. We've already got 'player' now.

It's defined here, we don't need it there. So, we're alright, we've initialized it, it's going to be set. Now, we need to change this a little bit. 'player' is a constant, so we can't actually store 'player', but we can say 'setPlayer()', and we can make it 'player + 1'. So, it's updating the value of 'player'. That's what 'setPlayer' does. And the convention is whatever variable you're storing, you put set on whatever it is. So, if this was 'state', we'd use 'setState'. If this was 'index', we'd use 'setIndex'. And then this is going to be in the initial value. So, we're going to have '(-1)' for the player. And now, we're going to set 'status' to the updated value of 'player'. Let's take a look at it running. So, I need to save that. Let's go back in here. Reload. Okay, 'Player' is '-1'.

That's good. Let's click on this, and we'll see we're here at the moment. So, 'Player' is '-1'. Now, we should have updated it, and we're going to update the 'status'. Okay, now let's continue, and we'll see if React updates. It does, 'Player' is now '0'. Now, let's click again. and let it go. 'Player' is '1'. So, let me get rid of that. So, you can hear me clicking. And we're updating the state, that state in a component. And that's one of the things that we need to do. So, take a look at the code and make sure you understand how to store the state. Now, the state is only initialized once. So, the first time this is rendered, React will call and use this initialization.

Thereafter, it's going to maintain the value of 'player', and it will update that. It won't re-initialize it, okay? So, the first time a component is rendered, 'useState' will be initialized. Every time afterwards that we call it. It won't be reinitialized. So, this will take the value of player an increment it by one. So, try that and make sure you can keep track of which player is playing. Now, how would we make sure that it is either 0 or one? Actually, why don't you make sure here that this is, 'the player needs to be 0' player 'or 1'. Why don't you try to do that? And I'll come back in the next video, and we'll see how to do it.

Video 4 – Control Component With OnClick Events

Okay, so let's continue with tic-tac-toe and see if we can build this game. So, we left it and I wanted you to be able to switch the players between 0 and 1. So, you remember that at the moment we have, 0, 1, 2, 3, 4, so it just keeps increasing. And what we need to do is we need to

change this so that it goes between 0 and 1. Well, I need to take this '(player +1);', and divide by '2' and take the remainder. So, here, if I divide by '2', and do mod '2', that will give me the remainder. So, this will either be an even number in which case dividing by 2 gives us 0, or an odd number, which will give me 1.

So, let's see if that works. So, reload this. So, we start with -1. Now, we didn't want it to be -1, but we wanted to start with 1. But at least it goes 0 and 1. So, let's go back in here. And what we want to do is we want to initialize this the very first time it's called. Let's initialize it to '(1);' so that X will be the player, the one playing X. So, let's, yes, now it initializes to 1. There we go. Okay, so we've fixed that problem. Let's take a look now at catching the events. So, what I wanted to do is on this click, I want to catch the event.

So, '(e)' is the event, it's coming in and it's the argument. So, we can catch it here. And we can say 'e.target.style.', now, let's see what we want to do. Let's change the 'background' color. I'm going to change it to ' "red"; '. Let's see if that works. Okay, save that. And let's go back and take a look. Reload. Interesting, Let's do it again. Okay, Let's just change one more thing. How about we change the size? Let's make it it was 600. Let's make it '400;'. There we are. Okay, so now when we click, it becomes '400;'. So, we can control the component via events such as clicking or onfocus or on onChange.

Video 5 – Add Squares As Child Components

So, let's continue with building our tic-tac-toe game. We've created our game-board, and that's the < div > there. I'm going to clean this up. But inside of it, we need now to get those blue squares, the tic-tac-toe squares. So, we're going to create a new web component called 'Square'. So, equals, and we need the fat arrow notation. And okay, now what we're going to do is here, just like we did with board, we need to 'return' something and we're going to 'return'. So, we put in brackets, and we're going to 'return'. We're going to make each of those blue squares a '< button >'. So, that's our web component.

Now, we're going to place that inside our ' "game-board" '. So, we've already got ' "game-board" '. So, inside here, we're going to have another '< div >'. And we can give it a 'className='. And we're going to give it ' "grid-row" '. So, we're going to create a row. And let's just stop there a moment. Because on that, inside that row we're going to put the squares. But I just want to check first that we've got a ' "grid=row" ' built for us. Let's go to, here we are. and now, we've got a grid-row. Just one. Now, we can put the squares inside this. Let's go back and do that. So, here we are. Now, we're going to put our squares inside the ' "grid-row" '. So, we need to put them inside that '< div >'.

I'm going to write a function to create these squares. And it's called 'renderSquare'. And all it does, it's a factory if you like, for these squares. So, inside here I can call '{renderSquare}'. And you'll see that it takes one argument. So, let me put that argument to '(0)', to begin with. And let's stop here. So, let's reload. Perfect. We've got one of our squares in there. So this, we've got now, two child components, if you like. We've got the grid-row and now we've got this square. But the

square is the one that's going to be important because that's where we need to put the crosses and the zeros.

Now, it's great that computing makes it easy. Once you created one square, we can create lots of them. So, I'm going to copy that. And I'm going to create three of them. Give them a number '(1)' and '(2)'. They're going to be the ids of the squares. Save that. And let's go back and check if we can. Let's reload. Perfect! We've got three squares. That's really pretty good. These are our first children squares of the game-board. So, now we've got a parent component and child components. Let's take a look how we can write on them their id. Let's see how to do that, but we'll come back and do that a little bit later.

Video 6 – Pass ID As State From Parent To Child

So, we've seen how to create our child squares. Now, let's see if we can give them an id. So, we notice here when we're rendering our square that we've got '(i)', and that's going to be '(0)', '(1)', '(2)'. So, we can give a property to the '< Square>'. We can just say, for example, 'id='. Now, we want to get the value of that '{i}' being passed in here. And that becomes a property of the '< Square >'. That's pretty good. Excellent! Excellent! Let's see if that works. Now, where are we going to write it? Well, let's see. We're calling '< Square >'. How do we pick up that here? We can pick up the properties. So, let's pick up the properties. And for example, we could write it in here. Let's see if that works. So, we're writing it as whatever it is on the '< button >'.

So, we're passing it in from the parent via the properties into the child. Let's see. Let's reload. You see we've got 0, 1, 2, pretty small, but they're in there. If I zoom in here, you can see they're right there written in the square. We need to maybe make them larger, but they're definitely there. Okay, let's just go back and do it slightly differently. So, here we had the properties coming in as an argument, but we can actually extract them directly if we use a bracket notation. So, we can just pull the 'id' out of the property like that. And then we can render it in here. And let's, so we'll write the '{id}' there. Okay, let's go back and re-render it. Beautiful. So that's one way of passing something from the parent. We're using the properties to get it to the child and we can manipulate it there.

Video 7 – Change Color Of Square And Track State

So, let's keep track of the state of each of these squares. So, here their IDs '0', '1', '2'. I want to keep track of, say, their color. So, let's figure out how to change their color if we click on the square. That's the first thing, and then we need to keep track of it. So, here we are. And we need to change the color of the square when we click on it. So, we need to go into '< button >', and we do 'onClick'. And now, we want to set that '=' to a function. So, we'll define a function here that takes the argument '(e)'. And okay, we'll put in here, what happens. So, '(e)' is the event, it's passed to us. So, when there's a click on the '< button >', so that's the square. We're going to get

this event '(e)'. And we happen to know that 'e.target' will tell us. That'll get us to the button that we've clicked on, this button here.

And what I want to do is change, for example, the 'style', and the 'background' color. And let me just change it to ' 'red' ', for example. So, let's just check that that works. So, I'll save that. And we go here, reload. It's blue. Click on it turns to red. And if we keep clicking it'll stay on red. Okay, so we know how to change the color of that. Let's maybe give it a few more choices of color. So, we've got that the color change on click. But let's, for example, give ourselves a 'palet' of colors. And we'll make it a array, ' 'red' ', ' 'blue' ', and ' 'green' ' maybe. Yeah. Okay, so we've got a 'palet' of colors. Now, let's choose, let's have a random function that'll get us a random color.

So, 'getRrandomColor' is going to be a function. Let's just put a 'const', and it's going to write, let's see. So, what we want to do is we want to return a random color from that 'palet'. So, I know that 'Math.random()' gets a random number between 0 and 1. And if I multiply it by '*3', it'll get a number between 0 and 3, our floating-point. And I want an integer, 'Math.floor' of all of that will get has an integer 0, 1, or 2. Okay? And that I want to pick out of the palet color. So, 'palet[' that will give me either palet 0, palet 1, or palet 2. So, let me 'return'. or in fact, I don't need to return. It will automatically do it if I just do that.

Okay, so it's like shorthand notation, but it's going to return that value. Okay, good. So, now we can get a random color. So, for example, down here, we could have 'getRandomColor();', okay? Let's just check that that works. So, let's go here. Okay, green random color, blue, green, blue, red. If I keep clicking. Okay, we're doing fine. We're getting random colors. Very nice. Now, let's see if that actually causes a re-render of, okay, so now we're getting that, but it's not being stored anywhere. Let's keep track of the state of this. So, the state of the square. To do that, we need to have a variable 'color' and 'setColor'. And that's going to be '='. So, we're going to do a use of 'useState', 'React.useState()'.

And okay, we need to set it to some initial value. Let's set it to, well, we'll just set it to '0'. So, we'll use color, and we'll, we could either put the actual color in there. Maybe we'll do that. Maybe we'll set the color, I'll put it to '('red')'. Let's put it to '('green')', okay. Okay, so now we want to keep track of the state. We'd say here, 'setColor()', and we do '(getRandomColor())'. Okay, and now here, we could just use color because we've set it above. Okay. Right, let's see how this works. So, start again. We're fine. Green, green, green, blue, green, red. Okay. So, it's working. And each of these squares is keeping track of its own color. We should check that, that there's no doubt that we're creating three squares, and each one is keeping track of its own color.

We kind of know that. How can we see that? Well, let's see. They all start off as blue. On the first click, why do they all go to green? Ah! Can you remember? Because it's the first time it calls this 'setColor'. It's going to initialize it to this. But once it's initialized, thereafter, it's not going to do that again. But we can tell that each of the squares has an independent state because they all go to green first. So, what I mean by that is that they all start as blue. This one now has called the useState and has initialized to green. Now, we can click on this, and it'll go to red, whatever. But these all go to green. So, this one's initializing its state. This one's initializing its state. Now, they're all keeping track of their own state. Okay, so we've got state and we're keeping track of it.

Video 8 – Pass State From Child To Parent

So, we saw how to move the id from the parent to the child. Now, what we need to happen is we're going to click on the square, so on the child. And somehow, let's first figure out, okay, the parent knows which player is supposed to play. Let's suppose we communicate that through the properties. Then we could write that on the square as who clicked. Let's just try and do that first. Okay? Because then we need to communicate back to the parent that we clicked. We need to communicate which square we are, whether we're 0, 1, 2, 3, whatever. But let's first, let's figure out whether we can get the player because the parent will keep track of the player. Let's pull that down to us. So, here we are. We've got the ID here, and the parent knows the 'player'.

So, how about if we put the 'player' into the '< Square>'? So, I type that in here. So, that's available. So, now in the 'Square', we need to pull out 'player'. Okay, and we'll put it in here. Let's go back and render it. Yep, we've got 1s everywhere, okay, so that it's easy. But the problem we've got now is we need to tell the parent that we were clicked on. Okay. Let's go and figure out where onClick goes here. So, we're going to have an 'oncClick' event. I believe it's a capital C. 'onClick=' okay. We're going to have to have, let's just, I just want to see, I just want to fire a function first. Let's put '{alert}'. Let's put, let's use the backticks. '(`I'm Square' what am I, '{id}`)'.

That'll do. '(`I'm Square' it needs to be [inaudible] '{id}`)'. Okay. This is just debug. So, this is, I just want to make sure that we're on the same page here. So, let's, that, I'm Square. Something's wrong here. Let's go back. I believe we need to be careful here. So, 'onClick' is expecting a function. Let's make it a function. So, here's a function. And this, we'll put it in brackets. Okay. So, and we've got a function here and in the body, we fire 'alert'. That should be, right. Here we go. Okay, that's good. I'm Square that. Okay. I'm Square 1. I'm Square 2. Okay. Try and think about now, how can we communicate back up to the parent that we were clicked on?

Video 9 – Component Lifecycle

So, let's explore a little bit more the lifecycle of components. So, we saw how to change the color of the squares. But let's step up now to the board level and see if we can control when these are being rendered. So, let's go and take a look at the code. So, here we've got the Square, and here we've got the Board. Let's take a look if we can stop the rendering of these so that we can flip them on and off. Let's put a button in here. We'll put a '< button >' component, and we'll put an 'onClick='. And so, let's click 'onClick=', let's call '{toggle}'. And we'll have that as 'Show/Hide Row'.

Okay, so, this is a function that we need to write. So, let's write that function up here. So, we'll have 'const toggle = ()' And we'll, let's define a variable called mounted. So, we'll toggle '!mounted'. And let's make that variable part of useState. So, we'll have something called '[mounted, setMounted]'. And we'll store that in 'React.useState'. And let's set it to '(true);'. So, it'll be just a true or false. And here then, let's, when toggle is stored. Let's do 'setMounted' to be '(!mounted);'. So, we'll just, if it's true, we'll make it false. And if it's false, true. So, we've got now 'mounted', and now, let's check if it's 'mounted'.

So, it's 'mounted', then this will be called. Let's copy this and we'll put it in all of them. Okay, let's see if this works. So, we'll reload. Now, we've got Show/Hide Row. We click and it's gone. We click again, it's back. Let's take a look at the IDE. So, I'm going to move this over, and here's the IDE. Let's mount it again. Let's start at the beginning. So, it renders all three. Now, we click this, and it unmounts Square 0, Square 1, Square 2. Now, flip it again, and it renders. So, that's quite slick that we can control whether things are mounted or unmounted. So, I can control all the children here. That's quite slick. So, why don't you program that and try that out for yourself?

Video 10 – Parent State Forces Re-render

Okay, let's go one step further and see if we can force a re-render. So, let's put another button. Let me just show you where we are. We're in the Board. Now, let's force another re-render by putting in a '< button >'. And let's call '{reRender}'. Okay, 'Re-render'. And let's do that by creating another useState. So, let's call this 'Random'. We'll just put a random number in here. And that will cause it to re-render because the state has changed. And let's put it '(0);', to begin with. And now, let's type in the 'reRender' method. So, here we'll just 'setRandom', and we'll do '(Math.random( ));'. Okay, that'll be enough to force it to re-render. Let's take a look. Okay, now we've got a Re-render button. Here, it's mounted and we see, let's scoot this over a moment in the IDE.

And you see we've rendered 0, 1, and 2. Now, let's hit Re-render again. So, you see it's unmounting and re-rendering. And this is forced because we're changing the state of the parent, and that's causing the re-render of all of these. And of course, we can, that would also force an unmounting and re-rendering. Okay. So, that shows us the life cycle of web components and what is going to be re-rendered and what isn't. So, if you change the state of a component, it will force a re-render of that component, but also of its children. So, it will go through all of the children and re-render them as well. So, that's what you saw here, that the squares are children of the board and they will be re-rendered if the state of the board changes.

Video 11 – Track Re-render With UseEffect

So, we've seen how to change the state of each square. I want to now track if it's being re-rendered or not. Is React re-rendering? Well, let's see. So, we can figure out if it's being re-rendered or not by putting in a 'useEffect'. We need to call 'React.useEffect'. And 'useEffect', we can pass it a function. So, we're going to pass it this function. And let's write out to the 'console' that it's being re-rendered, and we'll pass in the '{id}'. So, that'll tell us which square is being re-rendered. And we can also when 'useEffect' is returning, we can pass it a function as well, and we can specify that function as. And this will be called whenever it's unmounting this web component. We're unmounting the square and will again give it the '{id}'.

So, now 'useEffect' is called every time this is re-rendered. So, let's see the lifecycle of this square. Okay, so, we'll save that. And now, let's come here. And I'm going to, you'll see that 'Render' was

called for '0', '1', '2' because they were rendered. Now, let's click on this one. So, it's an 'unmounting Square 0', and rendering '0' again. If the value in 'useState' changes, it'll re-render. But you'll see it didn't change. Let's click once more. Yes. Now, it changed. Now, it changed again. So, every time this one changes, okay. This one, it was just initialized. Now, it's rendering 'unmounting square 1', rendering square '1'.

So, every time we change the state, it's re-rendering, it's unmounting, and then re-rendering. So, 'useEffect' lets us track when it's being unmounted. And we can take actions and the same when it's rendering. Okay, now, let's go back in here with 'setColor' here. Suppose I just call 'getRandomColor()' here. So, the difference now is we're changing the color by going into the DOM directly or into the styles and changing them. But we're not updating 'useState', okay? We're not storing the value. We didn't set color. I just want to show you this because you'll see that it doesn't re-render. So, let's go back in again.

Here we are. So, it rendered the first time. Now, we can change, but there's no re-rendering going on. So, React doesn't re-render if we can directly into the DOM. The Re-render is triggered when we change useState. So, here if we change the useState here, it will re-render. So, let's put this back to what it was. Now, with changing the useState, we're setting the color and it will re-render. Just check that. Go in here. Yes, unmounting, rendering. Here, same thing, 'Render 1', 'Render 2'. So, we can force unmounting and mounting, and that will cause React to re-render. Okay, so that's our kind of state and lifecycle, our first lifecycle of each of these components.

Video 12 – UseState Delay

So, I want to show you a tricky little problem that I've come across. So, let's take a look at our code. Because we've been setting colors here, for example, such as this is green. And let me just show you what happens when we're clicking the mouse button. So, here we are in 'Square', and we're doing 'onClick'. And we're getting a random color. And then we're using useState to set the color. Now, you'd expect that now I could use the useState to color here on this very next line to set the color of the square. But it doesn't quite work like that. Let me show you. If I go to the debug environment. So, I've stopped here just after 'setColor', I've stopped on line 15. Now, if we look at the random color, it's supposed to be ' "blue" '.

But if we see what the color here is, it's ' "green" '. So, what this means is that when we're using useState, there's a delay between when we set, when we fire 'setColor', and when the 'color' has actually been updated. So, you'll see here that the color is ' "green" '. So, this is a problem that if you like, this is an asynchronous 'setColor', and it may take some time for it before it's updated. And here, I'm using, I'm using '(col);' to set the 'color', and then I'll immediately using 'color'. Let's let this go. And we'll see that instead of being blue, the color is green. Now, if I click again, it's going to, let's go. It will set the color to blue. So, we've got to be a little bit careful in using when we set this in useState.

We shouldn't use useState immediately afterwards. So, what I need to do here is if I put 'col;' there, then everything will be okay. So, let's see it in action now. I'm just, let me save that and go

back here. We'll start again. Here we are. I'll click on it this time. And instead of green, It's now blue. It got the right color. Okay. So, be careful of using useState. Don't expect it to immediately have updated that useState. Okay. So, React is doing that asynchronously.

Video 13 – Passing Player From Parent To Child

So, we saw how we could pass the state from the child. So, here when I click upon the square, we could see that we could pass that state up and we could keep track of it. Now, let's look at the code. Because what we need to do now is to know the player that's playing this move and that's kept track of by the board. So, we need now to tell the child who's playing the next turn. Let's take a look. So, we figured out that the 'newState' would receive an object and would calculate the 'nextplayer'. And here, I've changed it a little. I've changed it now so that I'm returning the 'nextplayer'. I've set the player. Now, I'm returning the 'nextplayer'. So, what does that mean? Well, it means up here, we can catch the 'nextplayer'. It's now returning from 'newState'. So, now here I am in the child, the 'Square', and I'm catching the 'nextplayer'.

Basically, my parent is saying, hey, here's the next player. Okay. So, it's going to be an X or an O. So, let's figure out how to write that. So, let's keep track of '[status]'. So, what I mean by 'status' is whether I'm an X or an O. So, we'll do the same thing. We'll have a 'React.useState()'. And what would we put in that, we can put '(null)' for now. Now, what I'd like is I'm going to get a zero or a one for nextplayer. So, the zeros are going to be the Os and the ones are going to be the Xs. So, let me have a little array up here that I'll just call it the 'xo' array. And we will put in it, we'll have ' ["O", ' to start and an ' "X"] '. Okay. So, now we want to render that. So, we've got the 'xo', and we've got the 'status'. So, we're going to put either an X or an O into the status.

So, we've got 'nextplayer'. So, that tells me if it's an X or an O. In fact, we can write that in here we'll put 'xo'. And then '[next]'. Why don't we put '[status]' there, yes, because that's the next player? I mean, that's, I know we're next. So, we're writing out that. Now, we need to update the status. So, we 'setStatus', and we put it to '(nextPlayer);'. So, we're saving it and then we'll write it out. Okay, that seems reasonable. Let's check if that actually works. So, here we go. We'll call it. Okay, we start with null being written in there. Now, we get an O put an X in there, put an O in there. X, X, O, X, O. So, it's working.

So, now we've solved I think all our problems. We're communicating the child state up to the parent and the parent is keeping track of whose turn it is and is informing whatever square is being clicked on. So, remember we've got lots of squares. So, each of them is keeping track of itself, its own 'status', and its color'. It's quite difficult that if we try to pass, for example, the next player through the properties, it wouldn't work because each square would have a different property and they wouldn't be updated in the way that they're being updated here. So, here is all the magic in 'onClick' because that's being clicked on a square. So, the square gets the player from the parent.

At the same time, it sends its state. And here, I mean, we're just sending 'color;'. We can get rid of this later. We needn't have the colors keep changing. And then we set the 'nextplayer'. We set

the 'status'. Now, once the 'status' has been set, we could put a check in here in the 'onClick', and not update anything if this [inaudible] is already been occupied. But this, I think, solves all our problems. So, we start empty board and then we start filling it in. Okay, we've had quite a journey. I hope you've enjoyed it. We're nearly there. I'll have one more where we lay out the whole board and then we'll be able to play tic-tac-toe.

Video 14 – Tracking Total Game State

We've seen how to change the color of these squares. Now, the board needs to keep track of these changes. So, it needs to keep track of the state. We know what the state is down in the child squares, but at the board level, it doesn't know what the state is, so we need to keep track of that state. Let's take a look. So, this is the code at the moment. I'll just scroll through it so you can see. So, we're getting random colors and storing them locally in 'setColor'. But globally, we don't know what the color is of each square, and we need to keep track of that. So, the first thing we need to do is to put in a state here. Let's just call it '[state,'. That is the total state of the game. And we're going to keep track of it.

So, as things are clicked, we're going to add them to the state. And that's why I'm putting up an empty array here. So, it starts off as an empty array. Now, the question is, how are we going to update that? Well, let's define a function here called 'newState'. So, we're going to make it a function. We're going to pass into that function an object with '({id:id,' and 'color:color})'. So, we'll just call it an object. So, we're going to take, 'newState', will get an object. What we want to do is to call 'setState' with an array. Now, if we need to add to the array, we've got to be a little careful here. So, what we want to do is we want to take 'state', expand out 'state' and add to it this new object. That will do it 'setState'.

Let's print out the object just so, so we'll say 'adding'. And let's print out the 'state'. So, object. Let's do '${JSON.stringify(ob)}' the object. Okay? So, that will print that out. So, now we've set the state. Now, the thing is we need to call this from our squares. When we're clicking on the 'Square' component, we need to call that function. I need to get that function into 'Square', into the component 'Square'. So, here's the function. Let's see where we call 'Square'. Here, 'renderSquare' here. Now, we're already passing the 'id' in the 'player' to 'Square', and these are passed as properties of 'Square'. Let's pass this function. So, we'll call it 'newState'. And we're going to put in '{newState}' that function.

So, now we're passing it into the property. Now, we need to get it out of that property here. So, we'll 'newState', there we've got it, and we want to call it here. So, let's call it. So, here we call 'newState'. And we need to pass in an object with '({id:'. We've got 'id,' and 'color:col});'. So, let's put the 'color:color});' in there. So, when we click on the square, we're calling the 'newState' function, which going to take this object and put it into the 'state' variable of 'Board'. Let's try it. So, here we are. Let's reload. So, far so good. We can see here what's being called. So, let's click on that. So, we're adding to state {"id":0, "color": "red"}. So now, that's been added at the board level.

Let's click again here. {"id":1, so, this is square 1, "blue"}. Let's add this again. "red", once again, "green", adding "green". So, this is now keeping track of the state. Let's go back and let's print out the whole state. So, here we just did '(ob)'. Let's print out. Now, we've got to be a little careful that the 'state' might lag, but I just want to show you what 'state' looks like. So, here we're going to print out the whole state. And it's growing because we're taking the old state, taking everything out of that into a new array, and we're adding the object on the end. So, these will be a load of objects and we're adding a new one on the end.

Okay, let's take a look. So, here we are. Let's start again. So now, state is blank. Now, state has got "green". Now, it should have "blue". And as you see, it keeps growing. So, the state, now, there we go. So, you see the state keeps track of the "id" and the "color" adding state. Why? That's because. Okay. I was clicking on the 0, 1. Now, we've got "id" for that one, and we've got the "id": 2s as well.

Video 15 – Tic-Tac-Toe Final Refactor

So, we saw last time that we'd solved a lot of the problems of communicating between the parent and child. And the child then needed to communicate back up to the parent. And we solved that by passing a callback function from the parent down to the child, that when the child was clicked on child would fire that. So, we got to the state where we had these three working, and we had the Xs and the Os, The X would go first. And we're writing out the turn of the player. But let's go back and take a quick look at some of the problems. So, we need to figure out clearly, what does the board need to keep track of and what do the square need to keep track of. Now, clearly, this board needs to keep track of who has clicked where on the board.

So, we need to keep track of each. This number square 0 has an X, square 1 has an O, square 2 has an X. So, how can we keep track of that efficiently? We also need to keep track of whose turn is it to play next. So, who's the present player? Who's the next player? And we need to communicate both of those down to the square. Now, the square keeps track of the mark made on it. So, the X mark or the O mark. It needs to keep track of that. And also, we're keeping track of color, but that's not essential for the Tic-Tac-Toe game, but we'll probably leave it in for now. So, we can probably have a more efficient way of keeping track of the state. All we need to know whether it's an X or an O. And we're saying that Xs are 1s and Os are 0s.

So, we could have an array of nine and just fill in whether it's a 1, 0, or an O. So, we'd start off with an array of nine with them all now. And then if 0 was clicked on it, we put a 1 there. If 1 was clicked on, square 1, and we put a 0, et cetera. So, that would be much more efficient. And we need to keep track of the next player, whether that's a 0 or a 1. So, those are the board's responsibilities. Now, the square component needs to keep track of the mark. So, we'll keep track of whether it's a 0, or a 1, or a null. So, 0 would be an O mark, a 1 would be an X mark, and a null means it hasn't been played yet. And optionally, we're keeping track of the color red, blue, or green.

Now, we don't need to tell the parent about what color we are. A parent, in this case, doesn't care. That's not part of the parent's responsibility. So, we'll clean up our code a bit, and we need to figure out then who's the winner. Well, we're keeping track now of the state. There are nine squares. And so, we have state[0], state[1], [2], [3], etc. Now, we fill them in as state[0] would be a 1 here, and state[1] would be a 0. But an easy way to check is if we figure out all the winning combinations. So, here's, if these were all the same, if state[0], and state[1], and state [2] were all either 0s or 1s, then that would be a winning combination.

Same with [3], [4], [5]. So, we can loop over these lines if you like. Each one of these lines is a winning combination. And check if state, and we just put these numbers in. If state[0] = state[1] = state[2], then we know we've got a winner. And it'll be whatever symbol is in that. If they were all Xs, it would be they'd all be 1s. And so, we've got an easy check. And if we go to, let's just go to the code and take a look at winner. I'm going to give you this. It's, so assuming that our state is just an array of nine, or 0 to 8, then we pull out of the winning line. We're going to loop over each of these lines. Okay, so we just go to winning lines. So, we just take them as '[i]'. So, this happens to be a two-dimensional array, but we pick out a row by just picking out the '[i]'.

And then we pick out each of these numbers, 'a' will be the first number, 'b', the second, 'c', the third. Then we check if the state of that first number is equal to the state of the second, and the state the first is equal to the state of the third, and the state of a is not null, then we've got a winning combination, and this will be the winner. So, 'state[a];' will either be a 0 or a 1. And 0 would be the Os in winning, and 1s would mean that the Xs have won. Otherwise, we return null. So, that's how we check for a winner. Okay, so let's take a look at what we need to change here. So, the first thing is that here we are on the 'Board', we're only going to pass back the 'id' of the square. So, 'idOfSquare' is passed by the square.

Remember, the square is going to fire this function. We're defining it up in board, but we're passing it down to the square. So, now we've changed that. Now, what we need to do is we're going to change our state to an array of '(9)'. And we're going to 'fill' them all with '(null)'s. Okay, so that's that. So, now we need to change the state of that square because it's been clicked on. And we need to know who clicked on it. Now, here we're updating the player, and we should do it afterwards. So, I've moved that down below, and now we're updating the player. We're getting the old player, if you like, the player that's already been set in memory. Remember, we're storing this in 'useState', that we're storing the player in 'newState'.

Okay, so that's good. Now, we need to change the state. We're only need to, just, we've updated the state here with a '0' or a '1'. Now, we just need to set the state again. So, that's easy. Okay, that's good. And let's now, we need to return not the 'nextplayer', but the present 'player'. And this is, we're going to have to change this as well. Let's think about it. We need to, yeah, we're going to let 'thePlayer' be the present 'player'. And now, let's have a look. We need to change this to 'thePlayer;'. Yeah. And they'll be the 'present player'. Okay. So, that's good. And now, we need to move the 'status' because this doesn't have the scope to set status. So, let's fix that up here.

So, now what I need to do is I want to check the 'winner'. 'checkWinner', pass in the '(state);'. And then I need to say if the '(winner)' is not null. Then I need we've got a winner. And we need to

update the 'status = `Player`'. And we need to figure out that it's the winner. So, we've done that, and let's just check that I need to put 'status' in here. And that will update that '{status}'. Let's just check that we're okay here. Now, let's have a look in here. And I just want to check that everything's okay here. I might change the name of this array to 'XorO' array. So, it's a fixed array, doesn't change. 'XorO'. Because this will be that whoever's played. Okay, let's just check. And we've got one more thing to do because we've only got three squares here. We need to increase this to nine squares. So, we'll copy that and put in nine of them. And we need to change these. '(4)' and '(5)', and '(6)', '(7)', '(8)'. Okay, save it.

And let's run. Okay. Yeah, we have nine squares. We've got the wrong player starting. Let's go back and just check that. We need to set that one to a '(1)' because we want the X to start, and we've changed our logic a little. Okay, X starts, and we should have a winner by now. We had one problem here that we forgot to change 'newState' to just write back the '(id);'. We don't need to do anything else, and that should check if that works okay. Now, we'll be checking. We've got the right state for checking. Let's have a look. Yes, player one wins. Okay, once more, let's see if it's okay. Yup, player one wins. Okay, so everything's working, and we've cleaned up the code a little. Why don't you take out the colors so that we have one that starts off that's all blue. Hopefully, you've learned a lot.