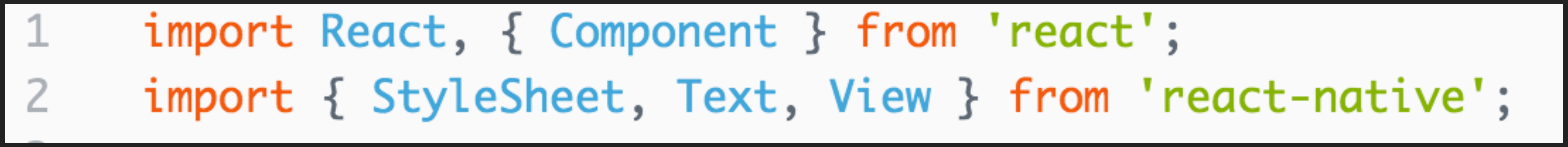
TIC-TAC-TOE IN REACT NATIVE

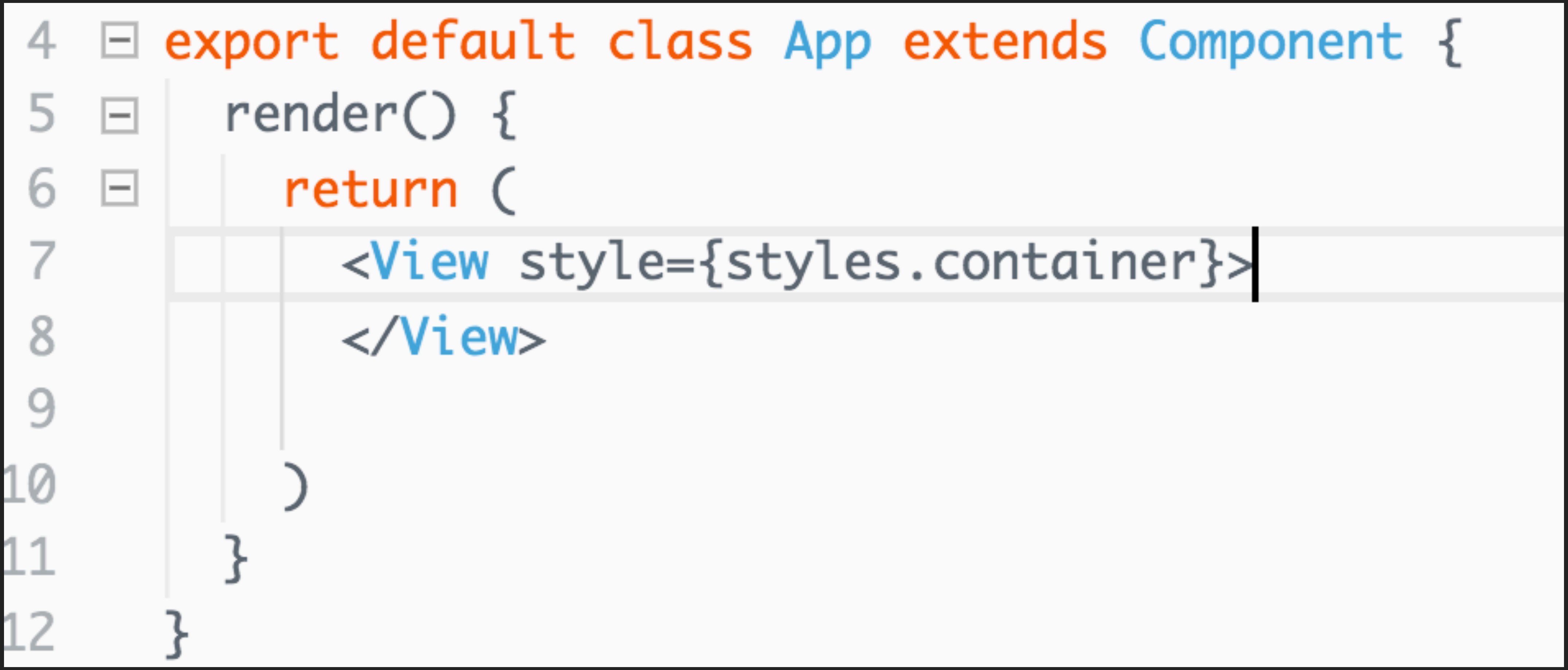
WHERE TO START

- Create or open up a new project in snack.expo.io or on your personal machine. Import your libraries and Components.



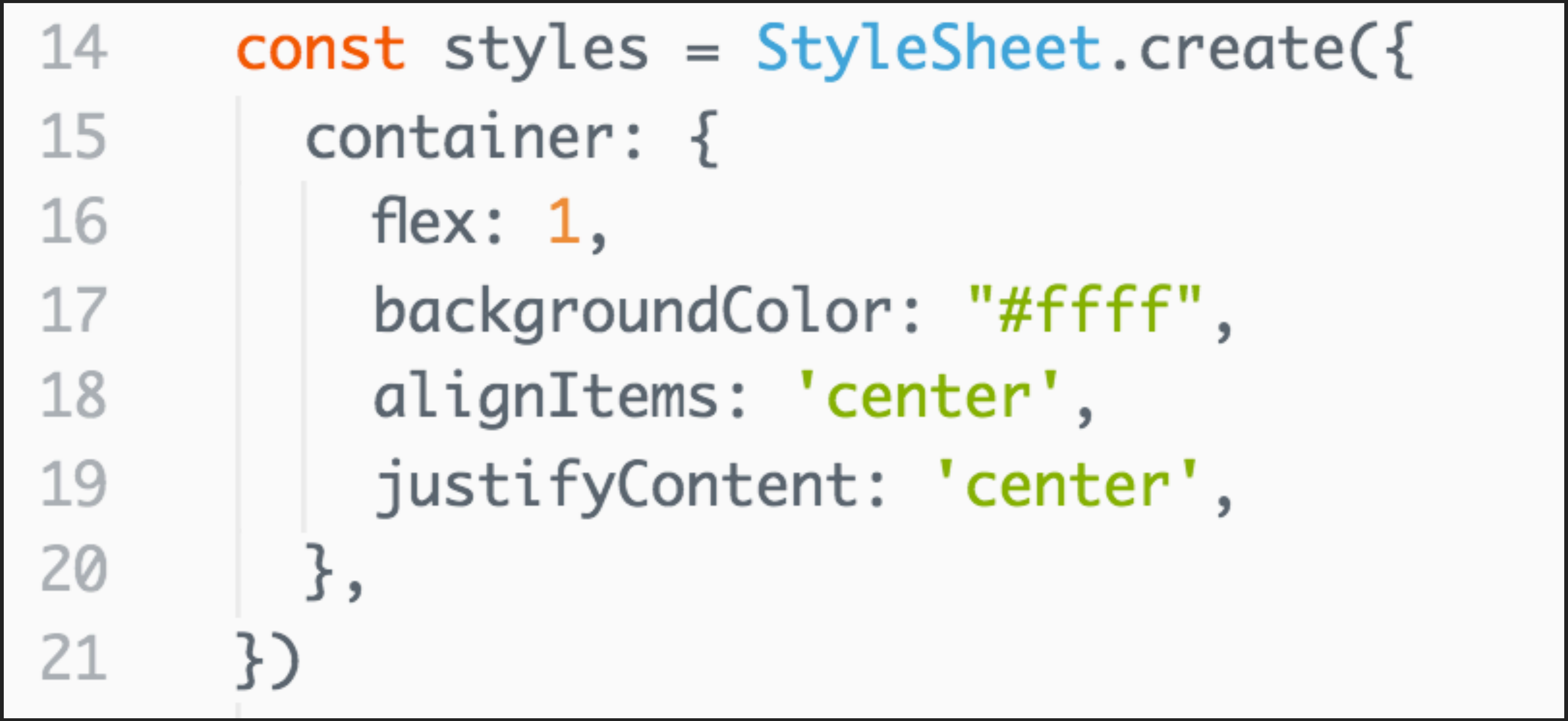
```
import React, { Component } from 'react';
import { StyleSheet, Text, View } from 'react-native';
```

- Create the base of our component. Here we're going to use a stateful class component.



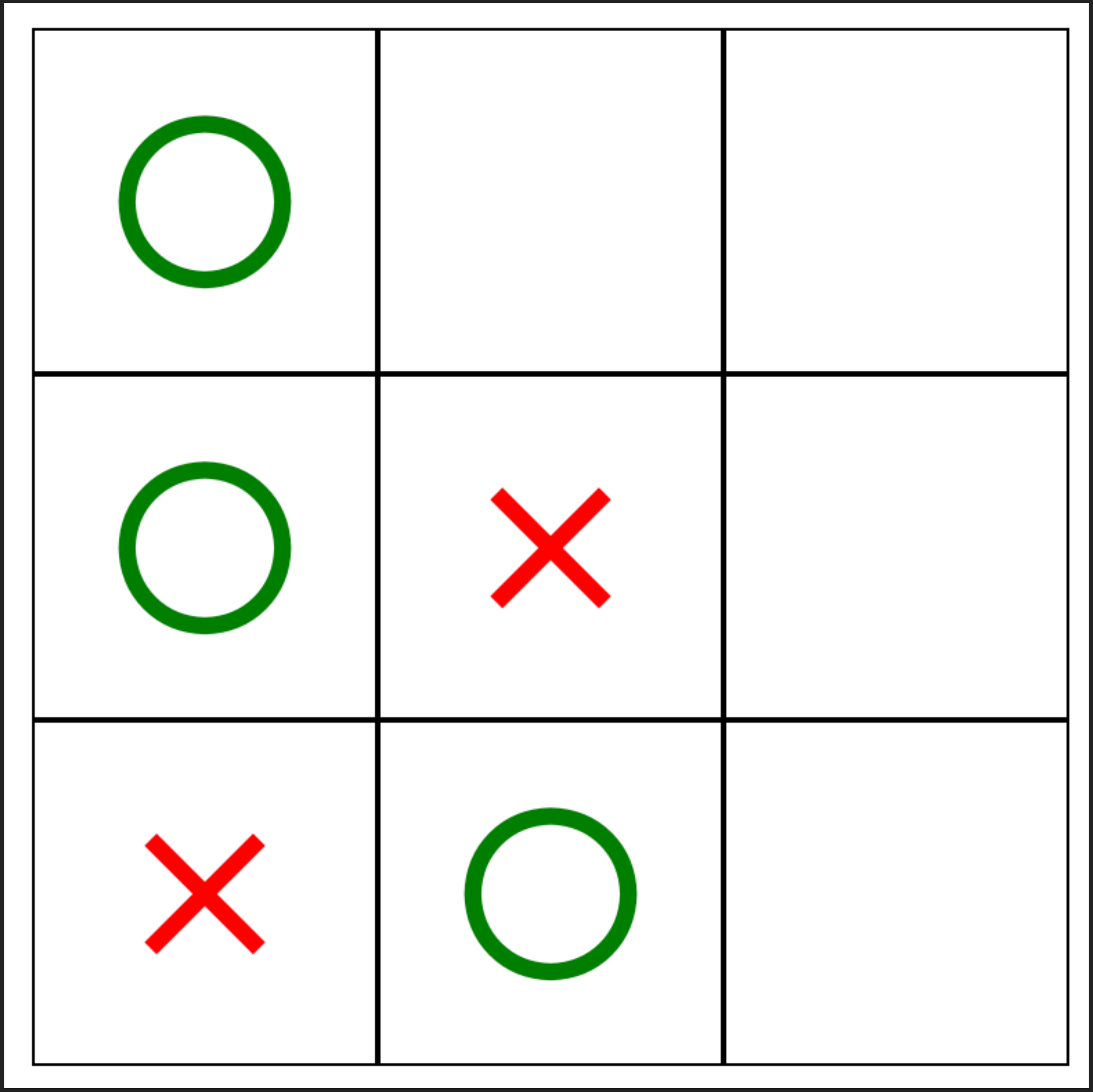
```
export default class App extends Component {
  render() {
    return (
      <View style={styles.container}>
      </View>

    )
  }
}
```

- Add some style to our app.

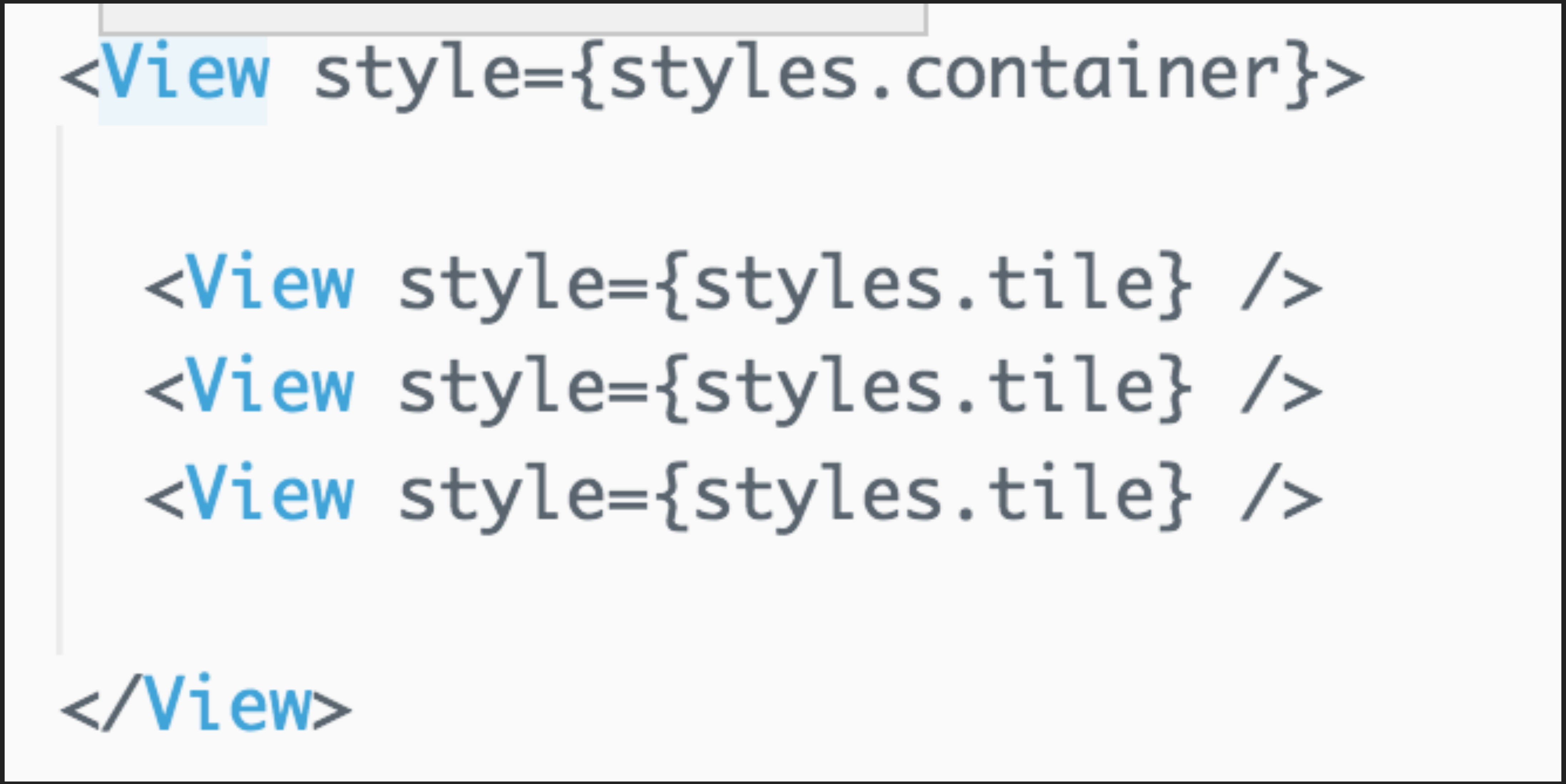


```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: "#ffff",
    alignItems: 'center',
    justifyContent: 'center',
  },
})
```

- Now let's think about how we’re going to build out our board. In tic-tac-toe the board is a 3x3 set of tiles.

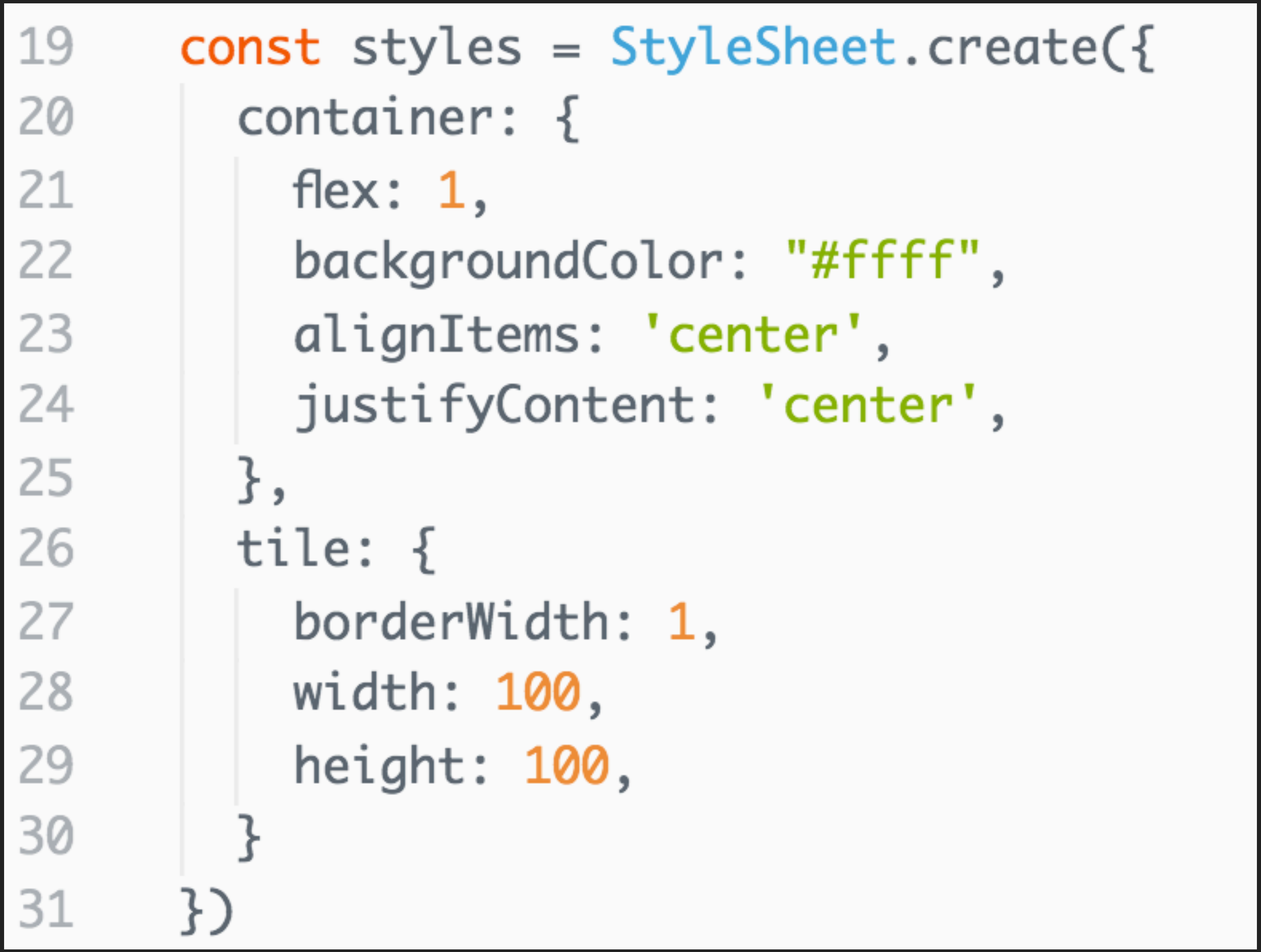


- First we start making our 3x3 tiles. Start by using the <View> component to build out our tiles.



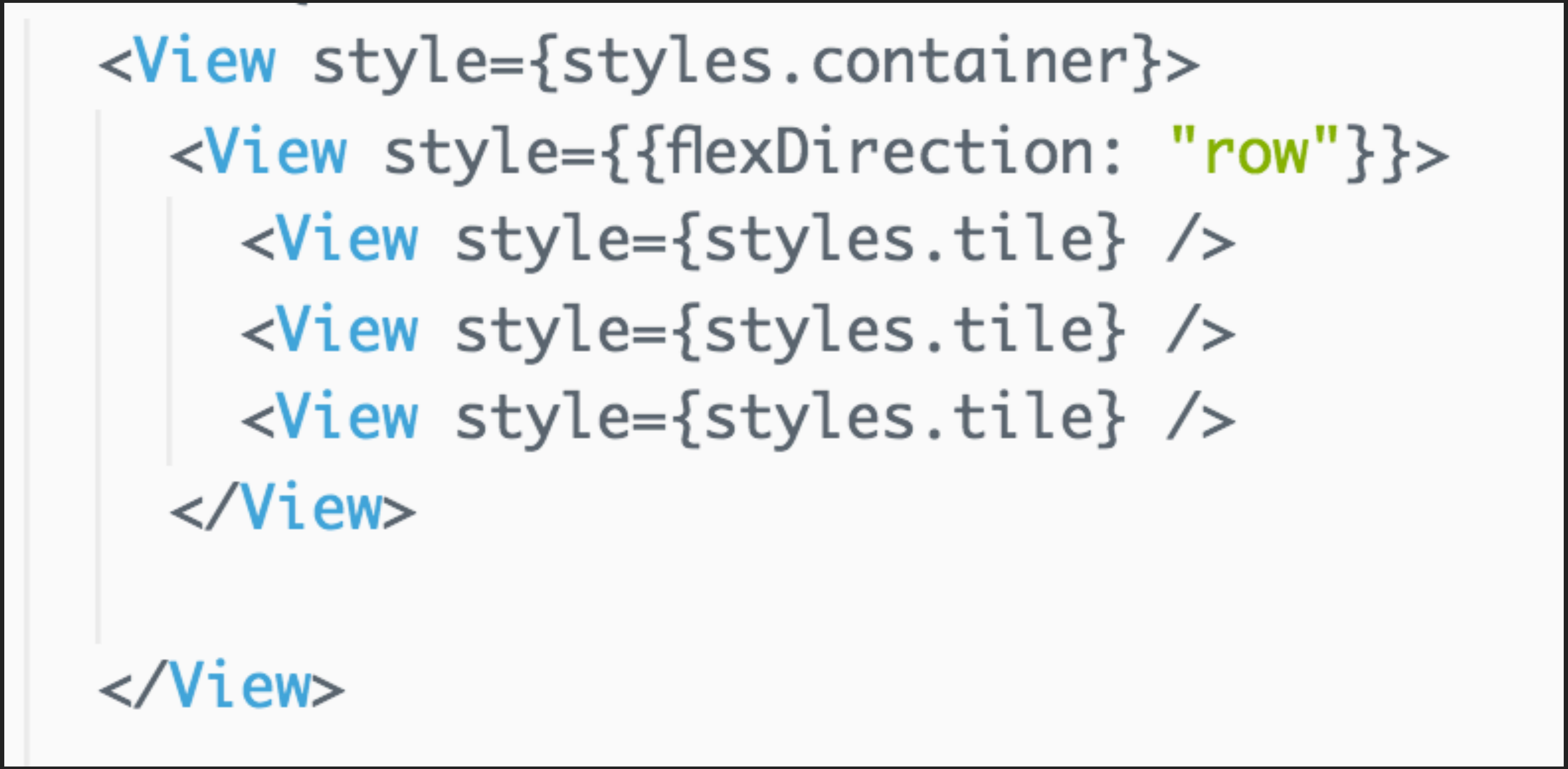
```
<View style={styles.container}>


  <View style={styles.tile} />
  <View style={styles.tile} />
  <View style={styles.tile} />


</View>
```

- Add some style for our tiles.



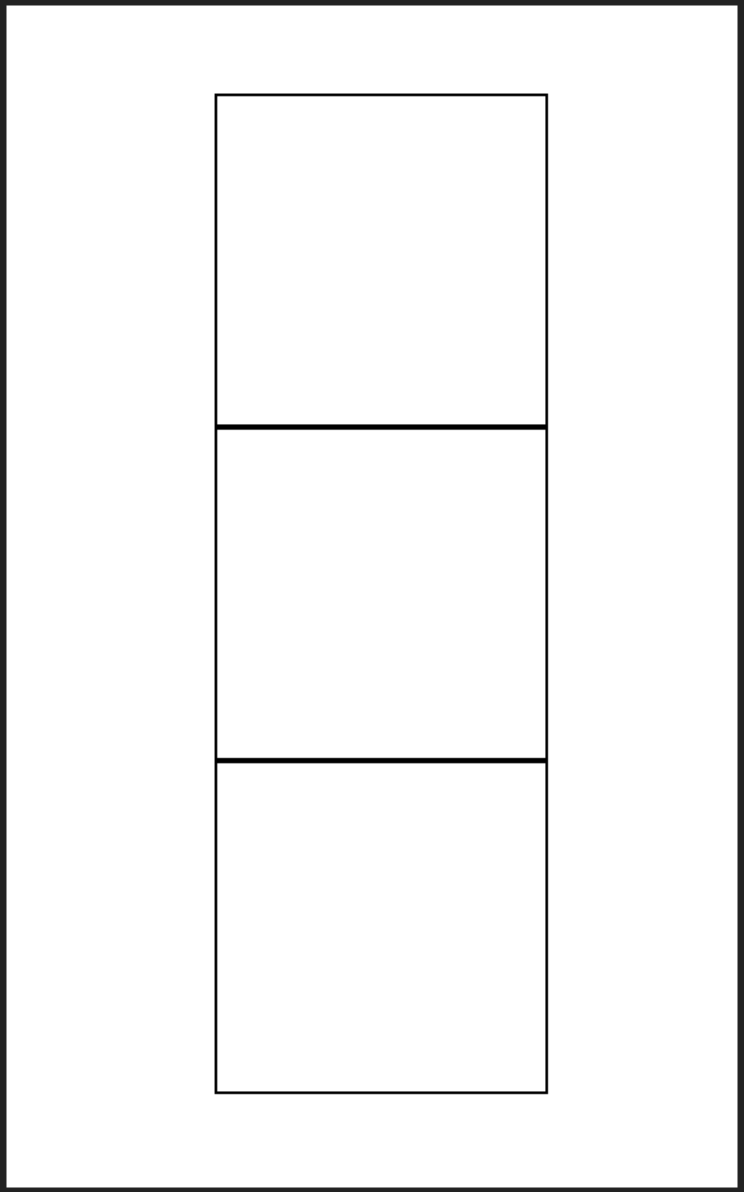
```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: "#ffff",
    alignItems: 'center',
    justifyContent: 'center',
  },
  tile: {
    borderWidth: 1,
    width: 100,
    height: 100,
  }
})
```

- To accomplish this – wrap your tiles in a <View> and add some inline styling to that component. Notice the syntax difference when using inline styling.

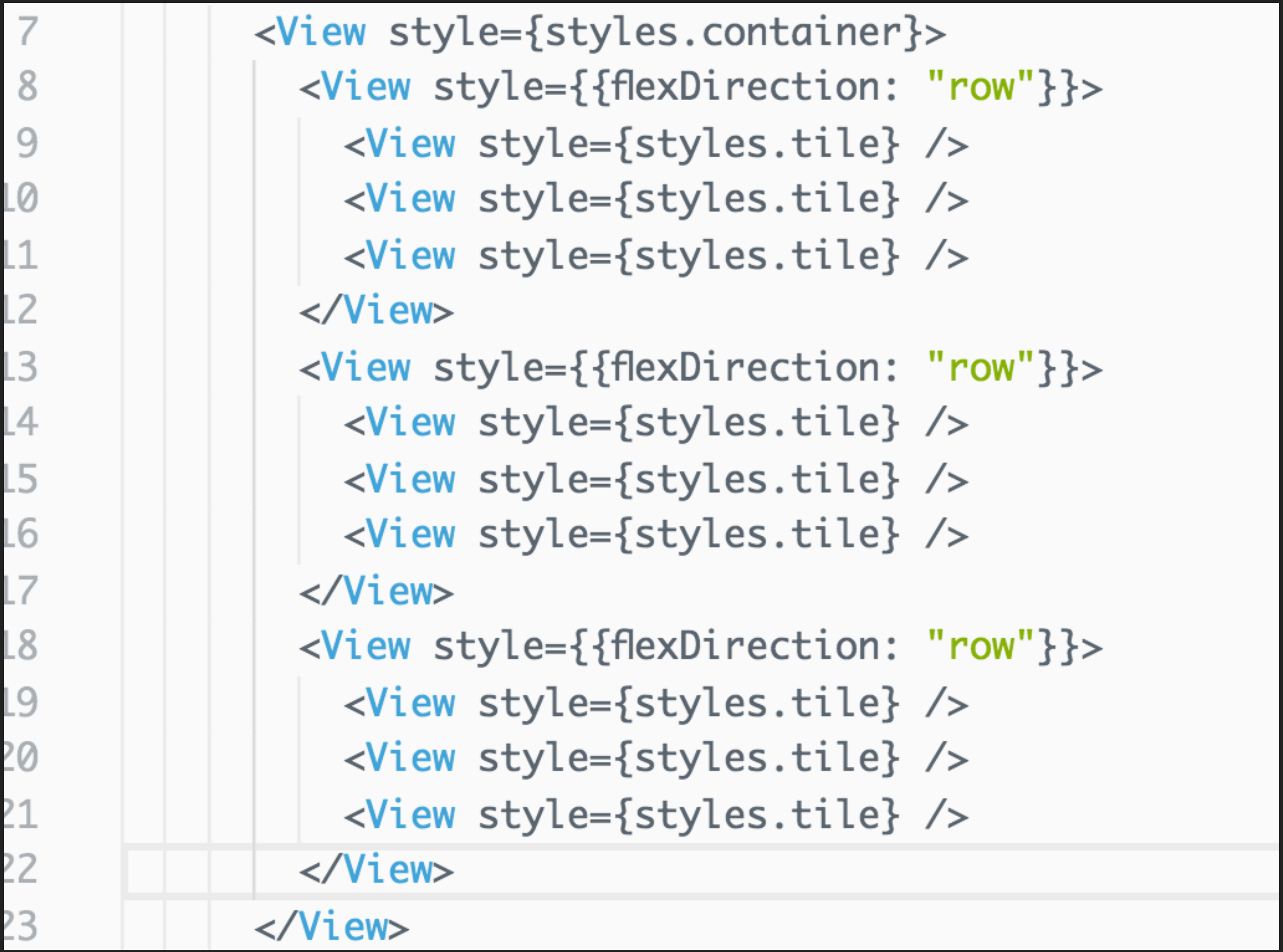


```
<View style={styles.container}>
  <View style={{flexDirection: "row"}}>
    <View style={styles.tile} />
    <View style={styles.tile} />
    <View style={styles.tile} />
  </View>

</View>
```

- AND BAM! If we look at our application – we can see that the boxes are rendering on top of one another. We want it in a row orientation.

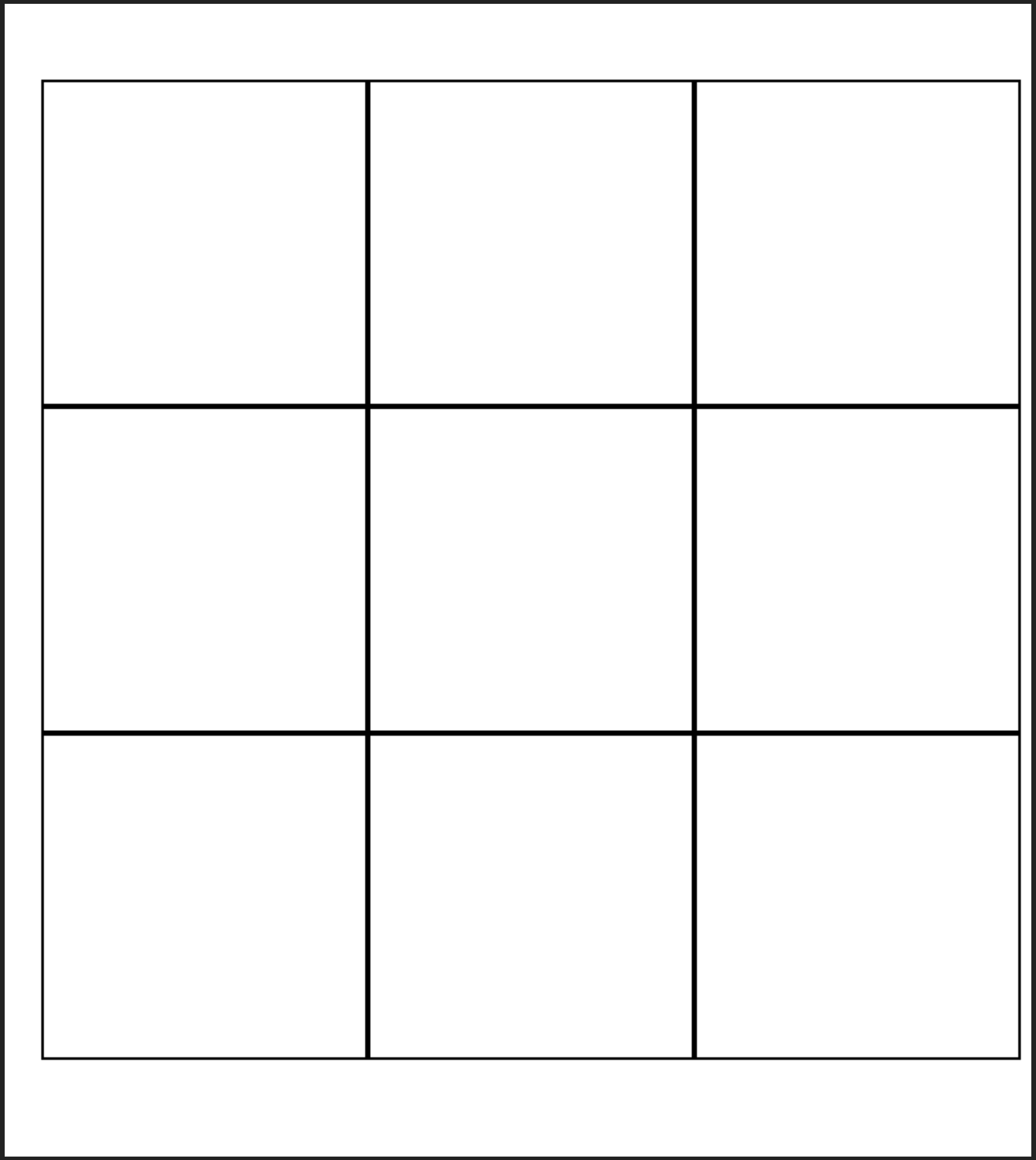


- Now that we have our boxes in a row. We need to create this two more times.



```
<View style={styles.container}>
  <View style={{flexDirection: "row"}}>
    <View style={styles.tile} />
    <View style={styles.tile} />
    <View style={styles.tile} />
  </View>
  <View style={{flexDirection: "row"}}>
    <View style={styles.tile} />
    <View style={styles.tile} />
    <View style={styles.tile} />
  </View>
  <View style={{flexDirection: "row"}}>
    <View style={styles.tile} />
    <View style={styles.tile} />
    <View style={styles.tile} />
  </View>
</View>
```

You should see a 3x3 board.



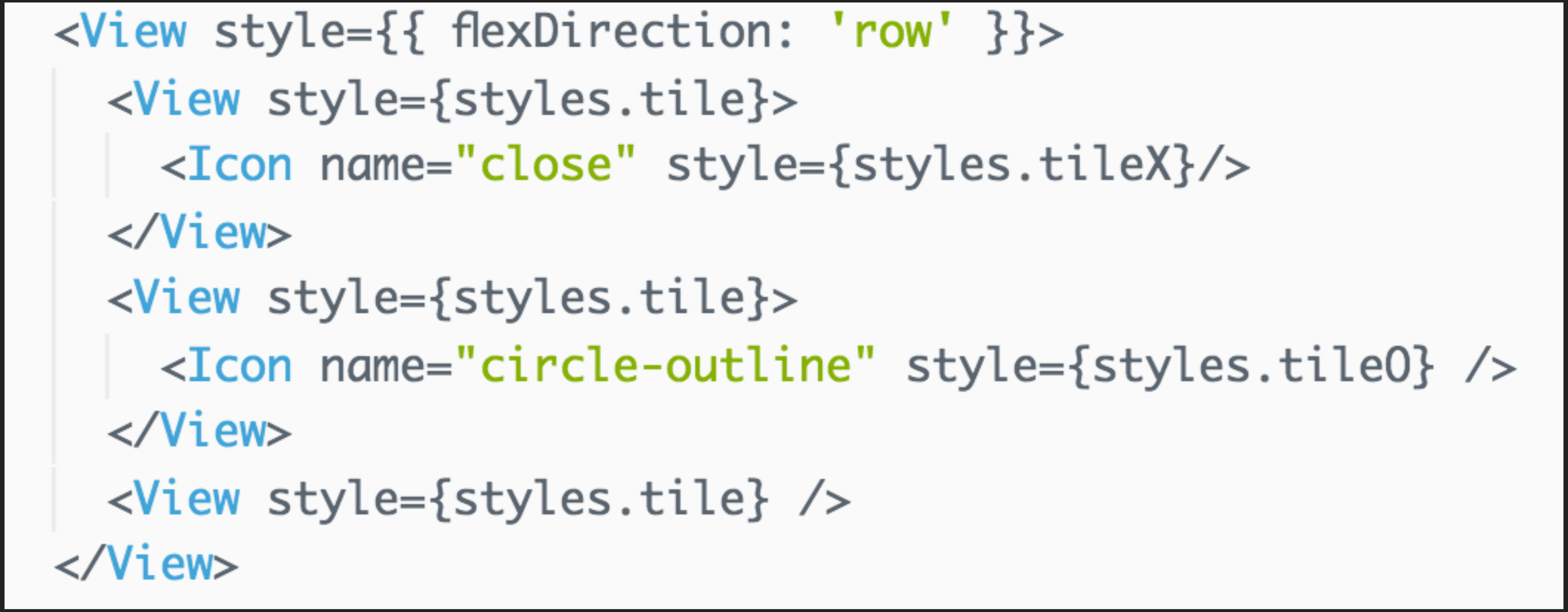
- Grab some icons from the MaterialCommunityIcons Component and alias it as Icon for the sake of brevity.

```
import { MaterialCommunityIcons as Icon } from 'react-native-vector-icons';
```

- Add some styling for both the X – and O tiles.

```
tileX: {
  color: 'red',
  fontSize: 60,
},
tileO: {
  color: 'green',
  fontSize: 60,
},
```

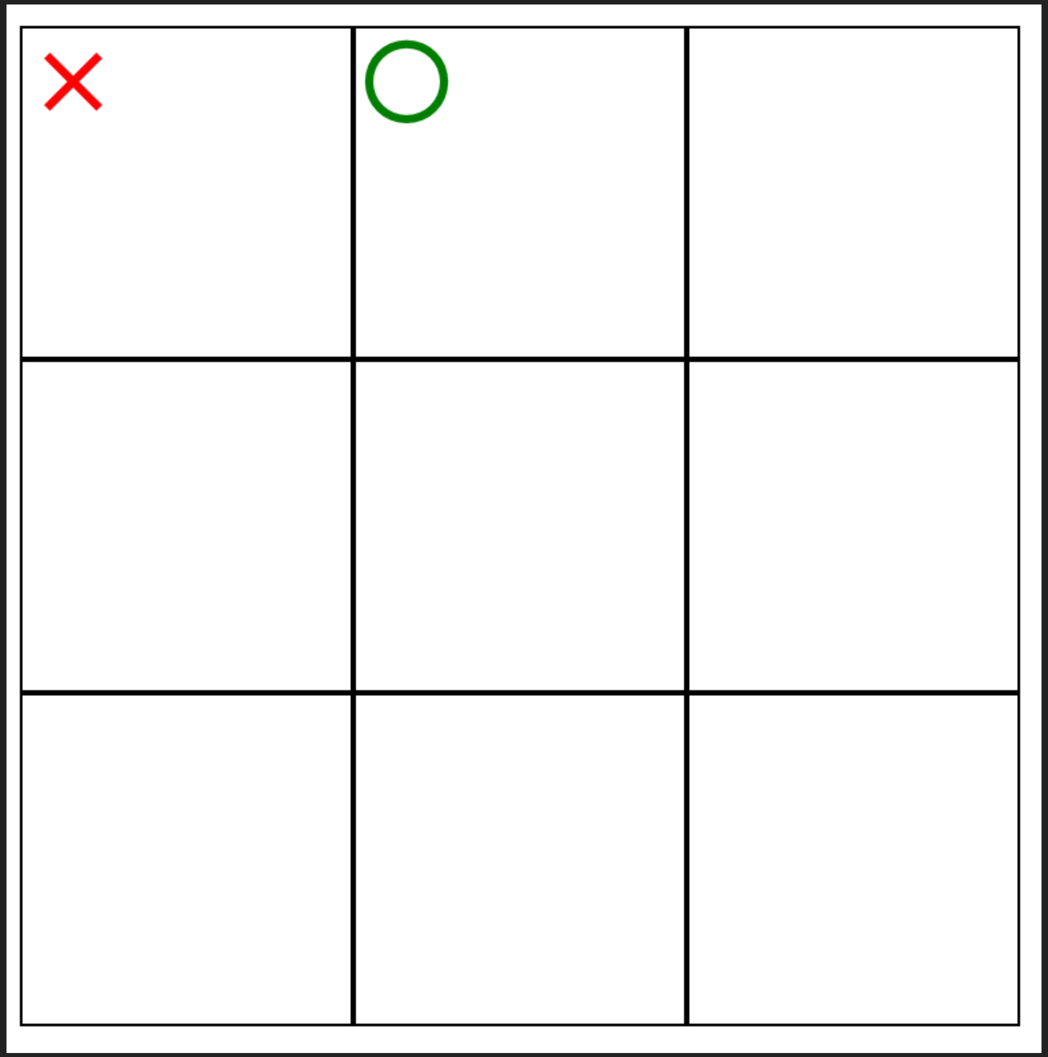
- Then let's test this out on our board.



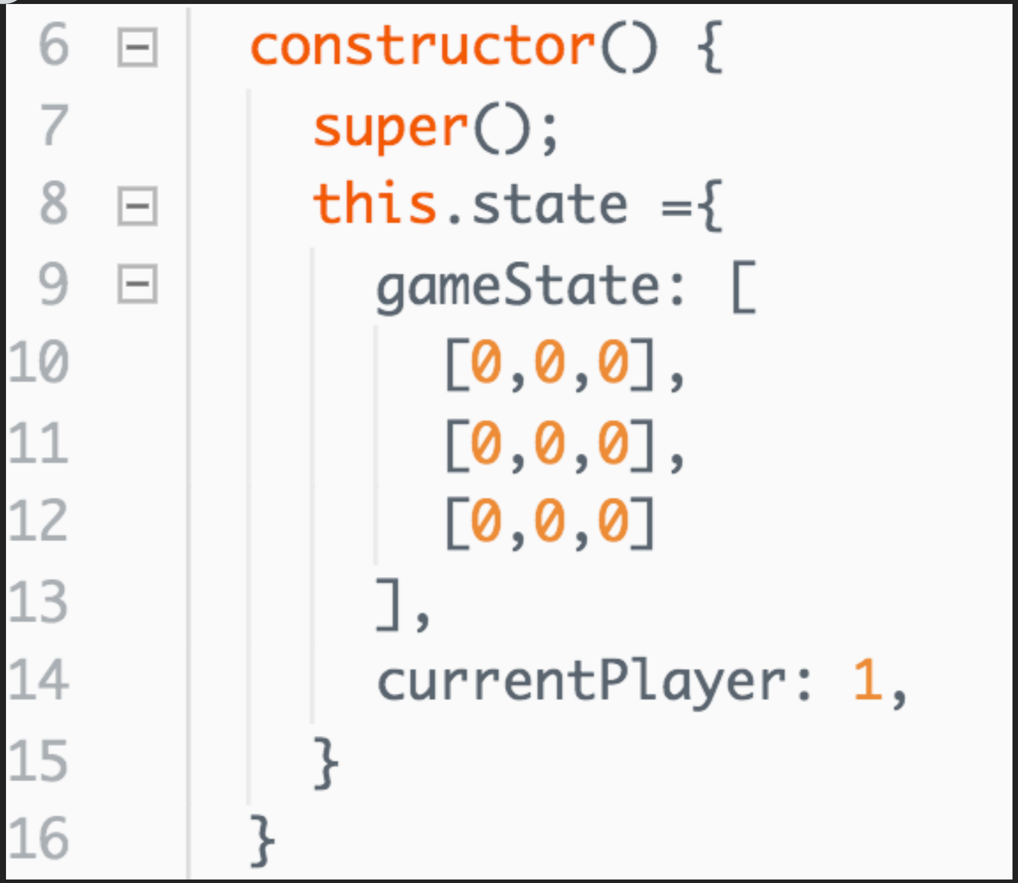
```
<View style={{ flexDirection: 'row' }}>
  <View style={styles.tile}>
    <Icon name="close" style={styles.tileX}/>
  </View>
  <View style={styles.tile}>
    <Icon name="circle-outline" style={styles.tileO} />
  </View>
  <View style={styles.tile} />
</View>
```

- Our Icon component takes a name that refers to the symbol you want to grab from the library.
- Link to MaterialCommunityIcons

TIC-TAC-TOE IN RN



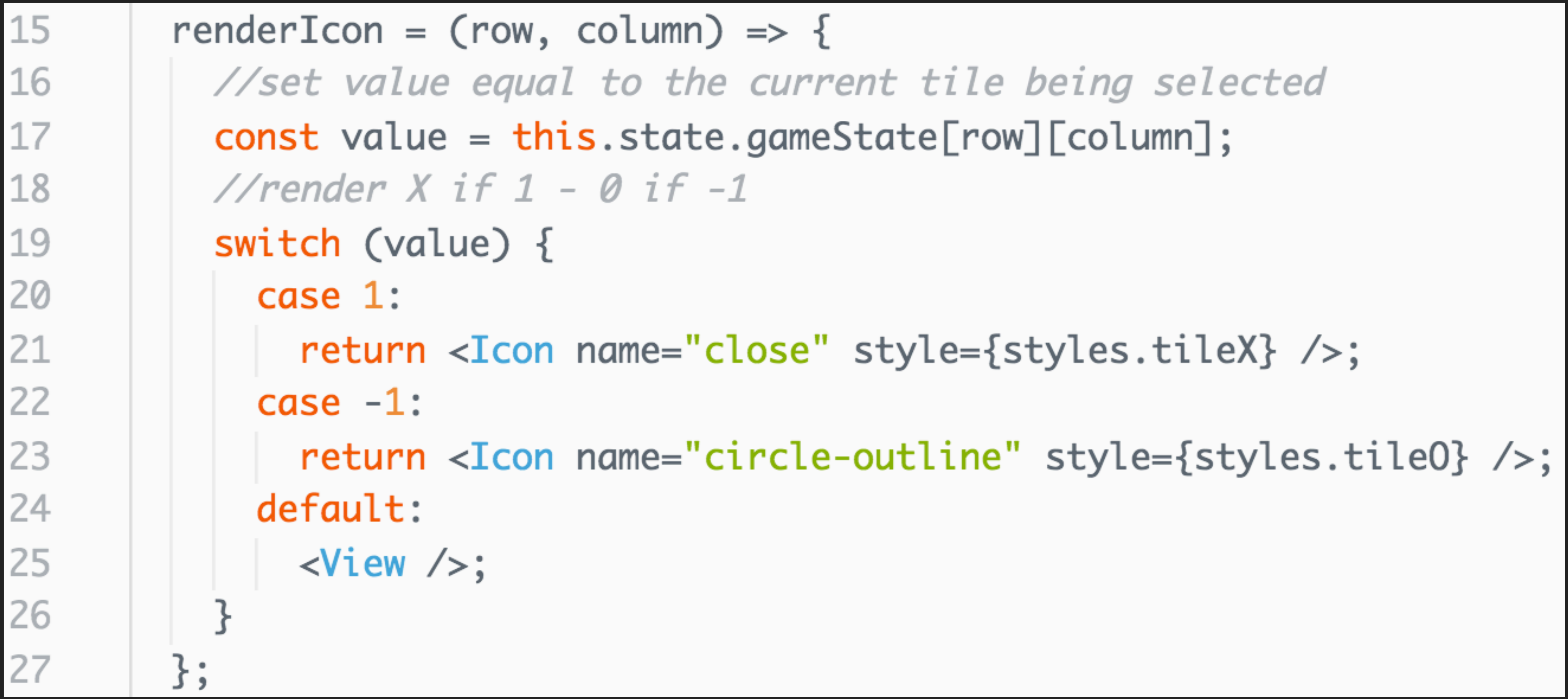
- Now that we have our program set up to accept X & O Values… let's start adding some functionality of how the game is going to work. First let's start by adding our state.



```
constructor() {
  super();
  this.state ={
    gameState: [
      [0,0,0],
      [0,0,0],
      [0,0,0]
    ],
    currentPlayer: 1,
  }
}
```

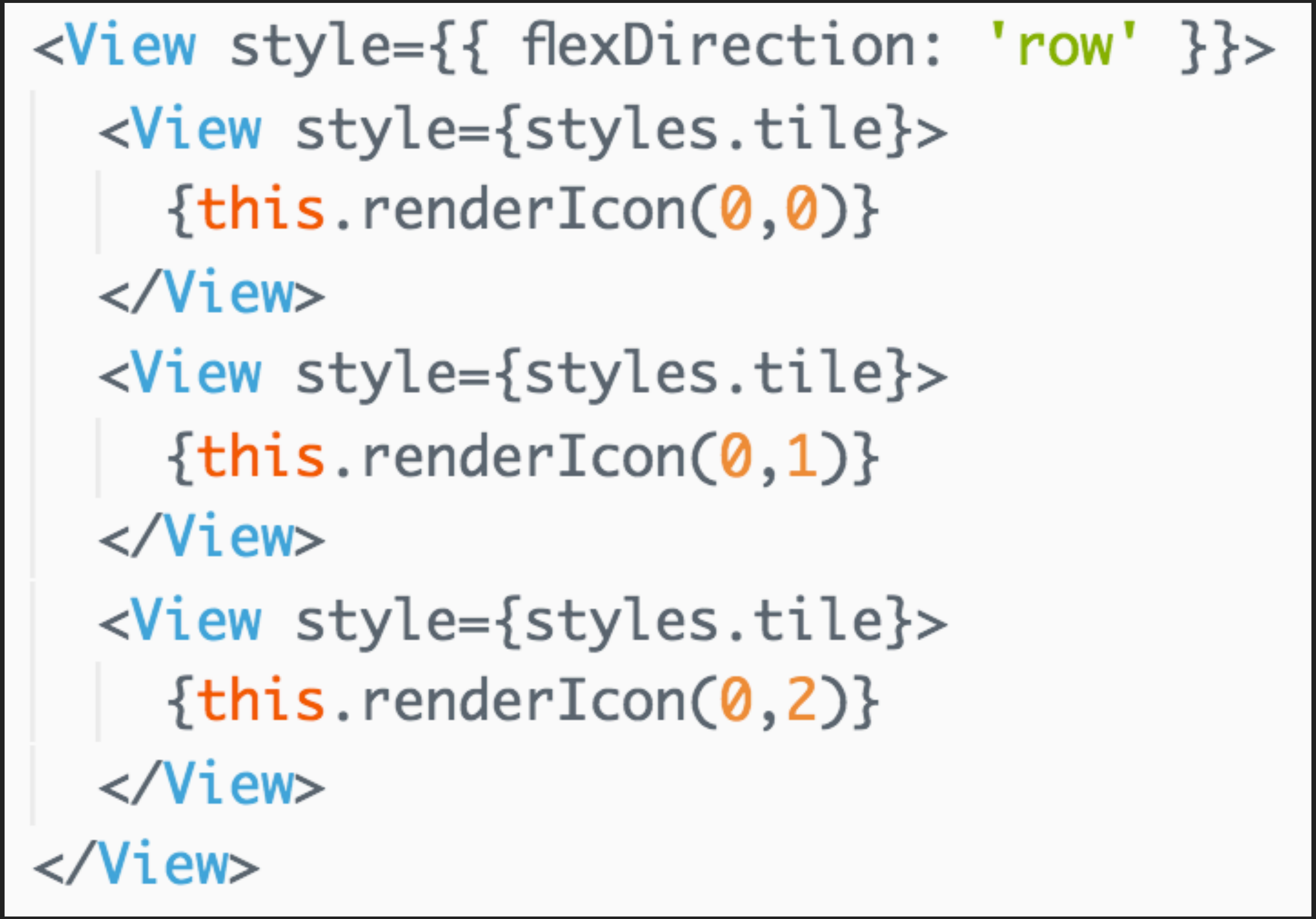
- The game state we're setting is how we're going to set values in our board. It's going to be a 2 dimensional array that is going to take the row and the column to set and X or O
- We also need to keep track of what players turn it is.

- Now let’s make sure that our board is rendering what is set to the current gameState. How we’re going to do this is create a function that will populate the board correctly. Let’s call this function renderIcon.



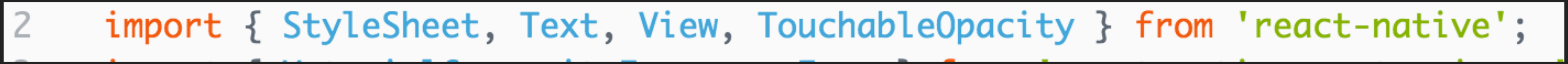
```
  renderIcon = (row, column) => {
    //set value equal to the current tile being selected
    const value = this.state.gameState[row][column];
    //render X if 1 - 0 if -1
    switch (value) {
      case 1:
        return <Icon name="close" style={styles.tileX} />;
      case -1:
        return <Icon name="circle-outline" style={styles.tileO} />;
      default:
        <View />;
    }
  };
```

- Once we have that method built out. We need to add in the correct values in our view to be represented by our board. You'll have to do this for every row.



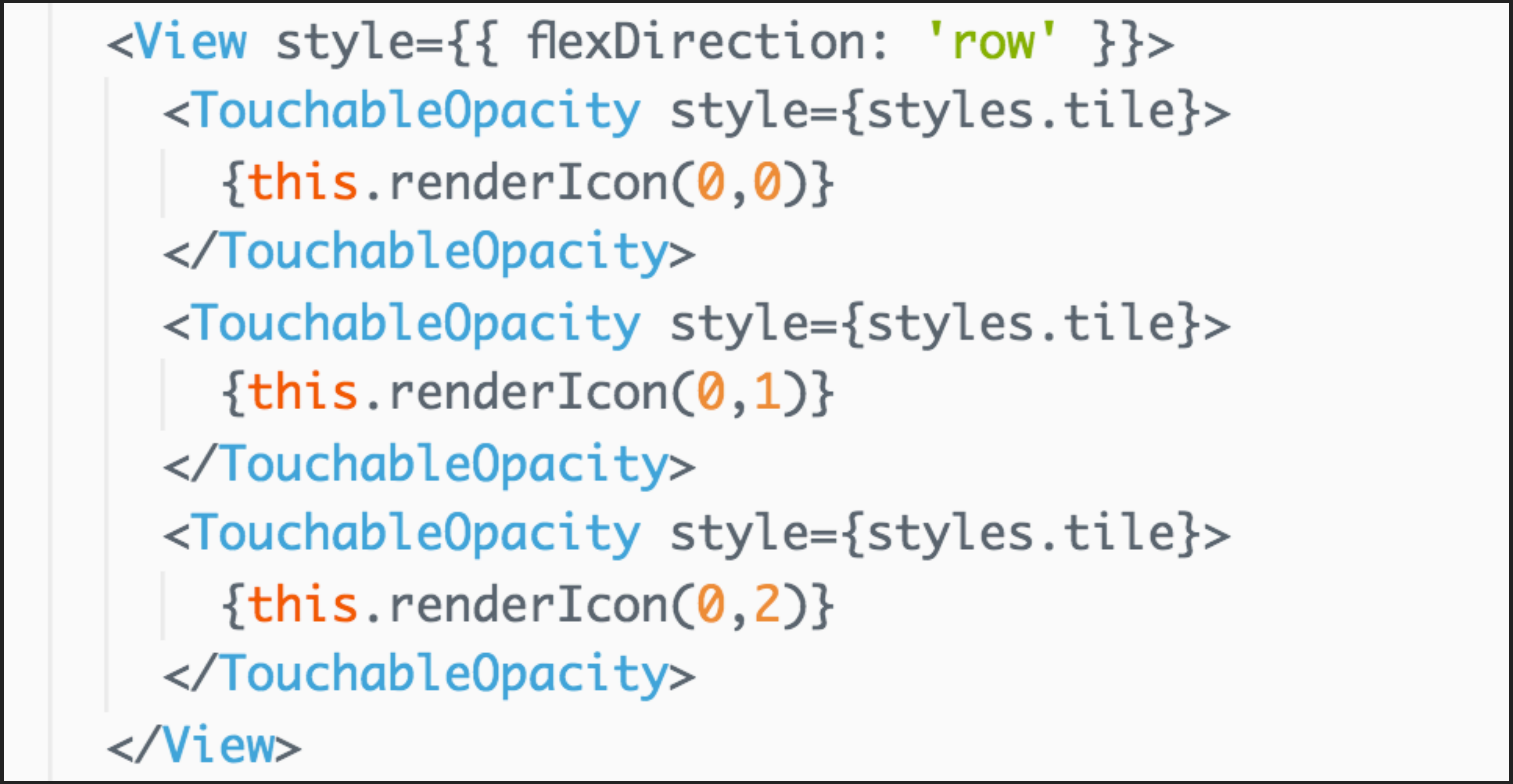
```
<View style={{ flexDirection: 'row' }}>
  <View style={styles.tile}>
    {this.renderIcon(0,0)}
  </View>
  <View style={styles.tile}>
    {this.renderIcon(0,1)}
  </View>
  <View style={styles.tile}>
    {this.renderIcon(0,2)}
  </View>
</View>
```

- Ok… now that we have our functionality for the correct icons to be displayed we need to add in the functionality that will allow our board to be touchable. So what we need to do is bring in a react native component called TouchableOpacity



```
import { StyleSheet, Text, View, TouchableOpacity } from 'react-native';
```

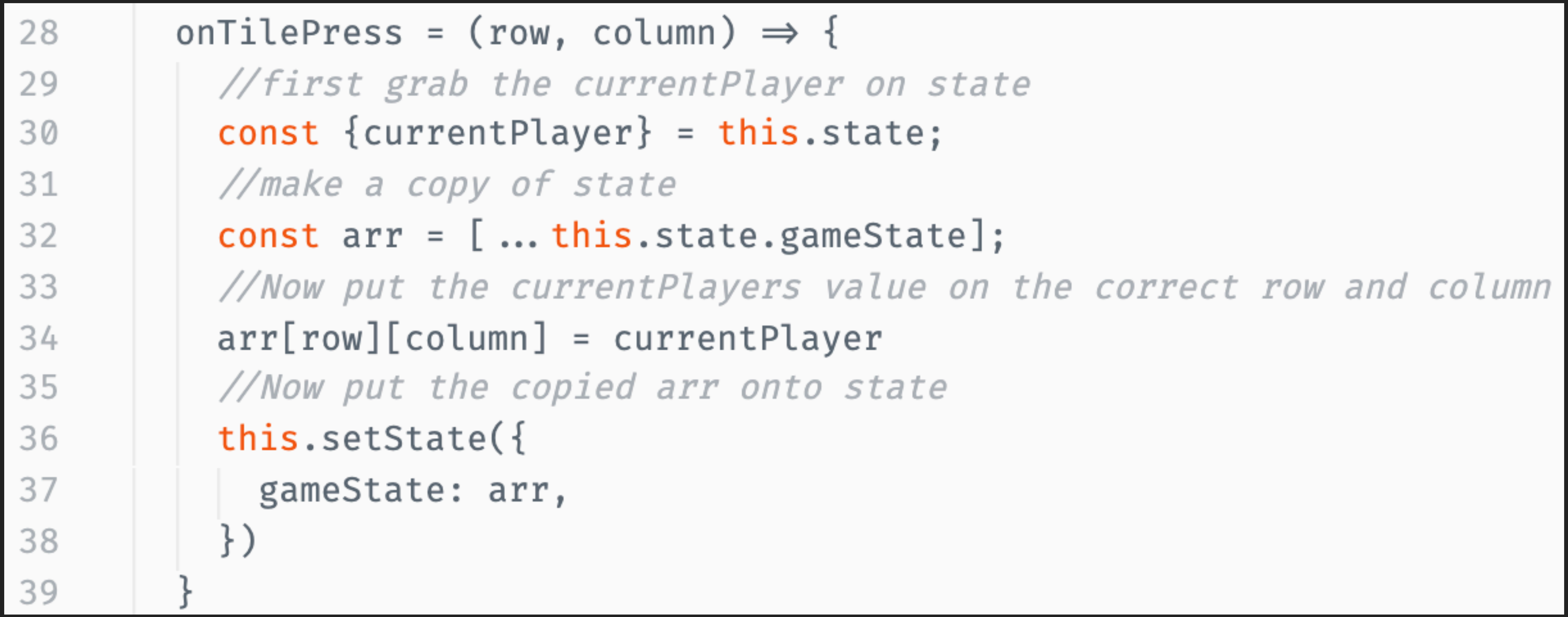
- Now we need to make sure that our board is able to be touchable. So we need to replace our <View> components that we are using to display our board with the <TouchableOpacity> component. Example:



```
<View style={{ flexDirection: 'row' }}>
  <TouchableOpacity style={styles.tile}>
    {this.renderIcon(0,0)}
  </TouchableOpacity>
  <TouchableOpacity style={styles.tile}>
    {this.renderIcon(0,1)}
  </TouchableOpacity>
  <TouchableOpacity style={styles.tile}>
    {this.renderIcon(0,2)}
  </TouchableOpacity>
</View>
```

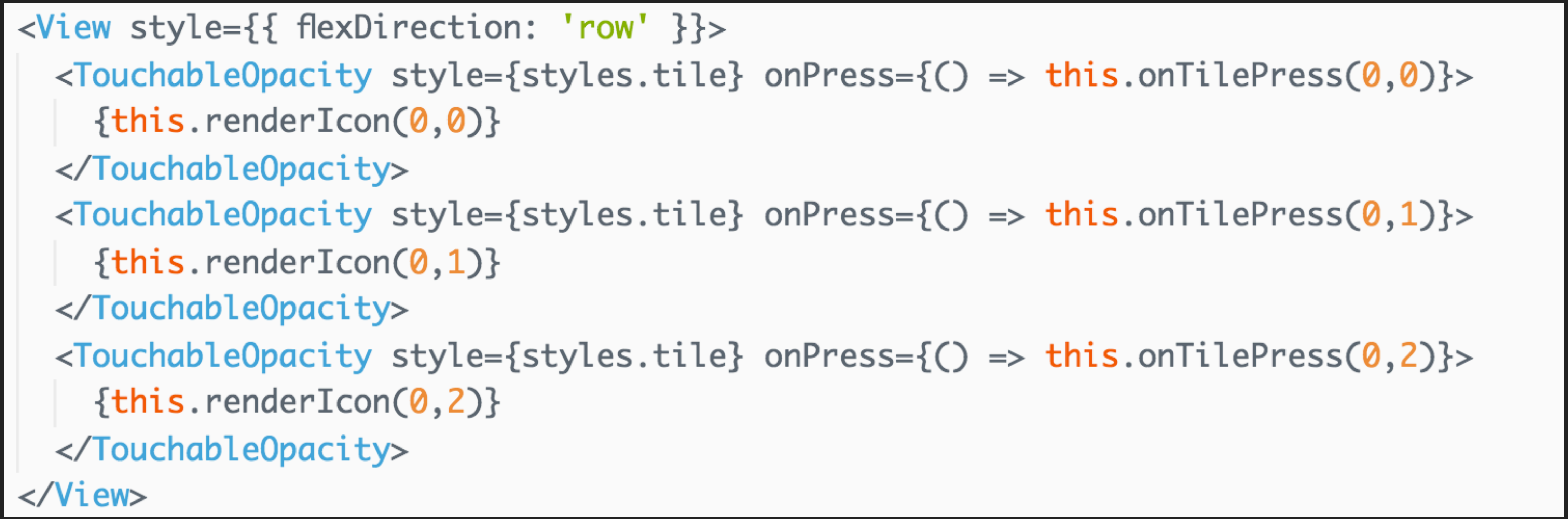
TIC-TAC-TOE IN RN

- Okay – Now that we have the ability to touch the board. We need to handle the event when that box is pressed. What we're going to do is create a function called onTilePress.



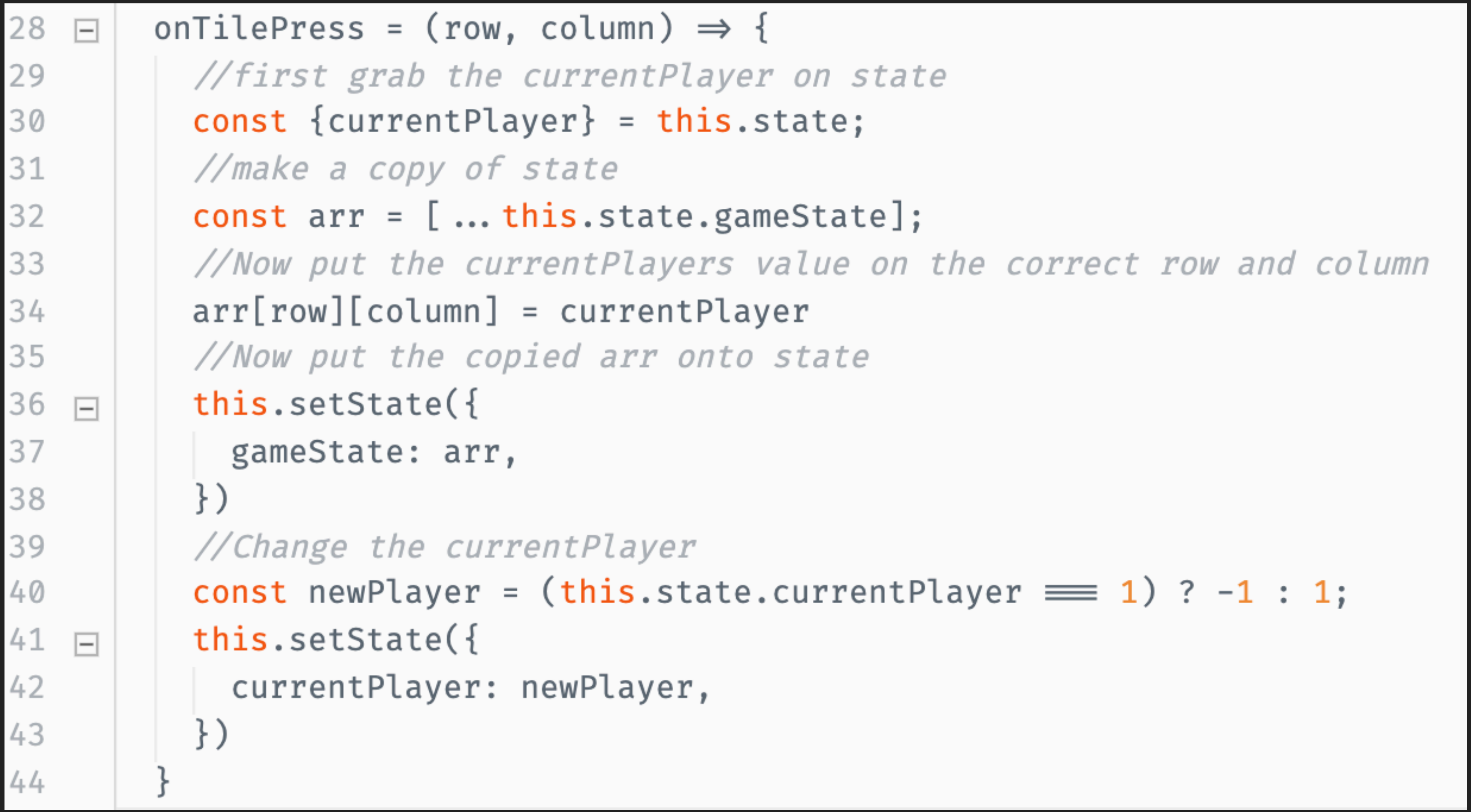
```
onTilePress = (row, column) => {
  //first grab the currentPlayer on state
  const {currentPlayer} = this.state;
  //make a copy of state
  const arr = [ ... this.state.gameState];
  //Now put the currentPlayers value on the correct row and column
  arr[row][column] = currentPlayer
  //Now put the copied arr onto state
  this.setState({
    gameState: arr,
  })
}
```

- Now that our function for handling the event of a Component being pressed. We need to add that onPress functionality to our <TouchableOpacity> component.



```
<View style={{ flexDirection: 'row' }}>
  <TouchableOpacity style={styles.tile} onPress={() => this.onTilePress(0,0)}>
    {this.renderIcon(0,0)}
  </TouchableOpacity>
  <TouchableOpacity style={styles.tile} onPress={() => this.onTilePress(0,1)}>
    {this.renderIcon(0,1)}
  </TouchableOpacity>
  <TouchableOpacity style={styles.tile} onPress={() => this.onTilePress(0,2)}>
    {this.renderIcon(0,2)}
  </TouchableOpacity>
</View>
```

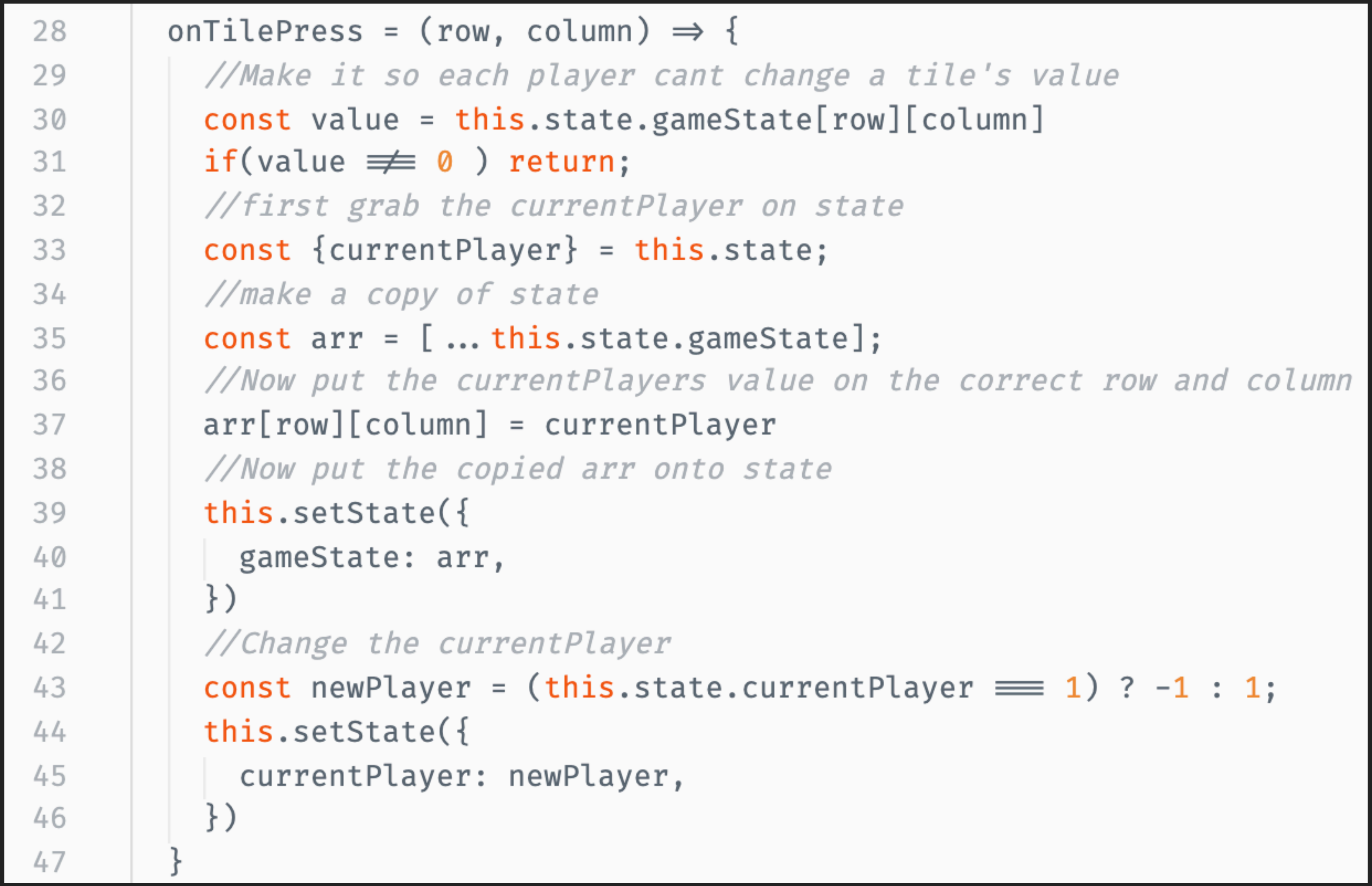
- Now that we have that working… We can tell that the board only displays X's when clicked. What we need to do is add to our onTilePress function that will swap between the currentPlayer.



```
onTilePress = (row, column) => {
  //first grab the currentPlayer on state
  const {currentPlayer} = this.state;
  //make a copy of state
  const arr = [ ... this.state.gameState];
  //Now put the currentPlayers value on the correct row and column
  arr[row][column] = currentPlayer
  //Now put the copied arr onto state
  this.setState({
    gameState: arr,
  })
  //Change the currentPlayer
  const newPlayer = (this.state.currentPlayer === 1) ? -1 : 1;
  this.setState({
    currentPlayer: newPlayer,
  })
}
```

TIC-TAC-TOE IN RN

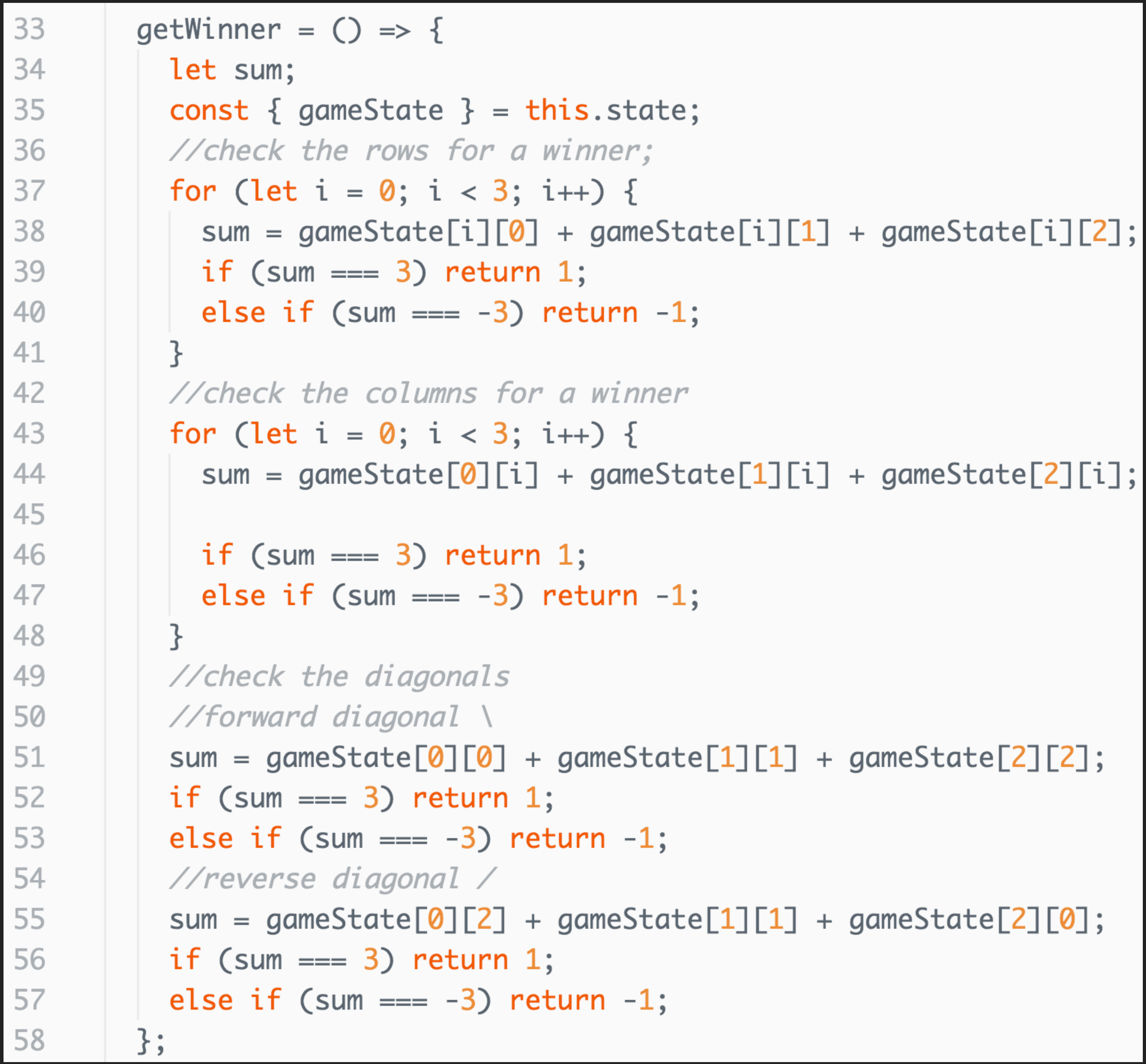
- Now we can see that it's switching the players but… we can click in the same box and change the value from X to O or vice versa. What we need to do is add some more logic into our onTilePress function, that will check if there is a value in the current array position - and if there is, do nothing.



```
onTilePress = (row, column) => {
  //Make it so each player cant change a tile's value
  const value = this.state.gameState[row][column]
  if(value !== 0 ) return;
  //first grab the currentPlayer on state
  const {currentPlayer} = this.state;
  //make a copy of state
  const arr = [ ... this.state.gameState];
  //Now put the currentPlayers value on the correct row and column
  arr[row][column] = currentPlayer
  //Now put the copied arr onto state
  this.setState({
    gameState: arr,
  })
  //Change the currentPlayer
  const newPlayer = (this.state.currentPlayer === 1) ? -1 : 1;
  this.setState({
    currentPlayer: newPlayer,
  })
}
```

- Now that we have our game board functionality working correctly. What we need to do is program our application to figure out a winner. So what we're going to do is make a new function called getWinner. Work on this for a while, and try to think about how to check for a winner based on how our game is set up.

TIC-TAC-TOE IN RN



```
  getWinner = () => {
    let sum;
    const { gameState } = this.state;
    //check the rows for a winner;
    for (let i = 0; i < 3; i++) {
      sum = gameState[i][0] + gameState[i][1] + gameState[i][2];
      if (sum === 3) return 1;
      else if (sum === -3) return -1;
    }
    //check the columns for a winner
    for (let i = 0; i < 3; i++) {
      sum = gameState[0][i] + gameState[1][i] + gameState[2][i];

      if (sum === 3) return 1;
      else if (sum === -3) return -1;
    }
    //check the diagonals
    //forward diagonal \
    sum = gameState[0][0] + gameState[1][1] + gameState[2][2];
    if (sum === 3) return 1;
    else if (sum === -3) return -1;
    //reverse diagonal /
    sum = gameState[0][2] + gameState[1][1] + gameState[2][0];
    if (sum === 3) return 1;
    else if (sum === -3) return -1;
  };
```

- Now we check for a winner every time that the tiles are being pressed. Add this functionality into our onTilePress function.

- Sweet it works.
- Edge case:
    - Make a function that will reset the game.
    - Add the logic for when the game is a tie!
    - Display which player is currently gaming.
    - Fix the bug when you start a new game - sometimes its player 2 turn.
    - Fix the styling so the X & 0's are centered correctly.