Portfolio 1
Tic Tac Toe with Facebook's React javascript library

Ben Meeder and Noah Eigenfeld

Table of Contents

Overview

In this Portfolio assignment we used Facebook's React architecture to build a client side Tic-Tac-Toe game. The React architecture is a javascript library that is used to build user interfaces. The main idea being the use of self contained elements that hold the styling, actions, functions and elements all in one class. These elements can then be treated as normal elements. Another benefit to React is the JSX language. JSX is a preprocessor that adds XML syntax to javascript. It helps to make React much more elegant and allows variables to be directly inserted to HTML element attributes.

For the creation we experimented and used a node.js project that allows the app to be compiled down to much faster binary. It also serves as a package manager so that React could be simply integrated into the project. Since React is a standalone package we could have kept this as simple JS files and just used link tags to import the CDN from Facebook. However why not step out of the box and use a package manager. If this were to be a larger product that would be a better way to go about this anyway so that any other modules needed could be very easily added.

We learned that creating a README.md for project is very essential to making it easy for other developers to contribute to this code. Also making a project modular and expandable allows for elements to be easily created making the maintenance of this app very easy. We concluded that once one becomes familiar with React it is very easy to quickly write web apps and make them look good. This app is also very portable as the components can be dropped into any other React application without any extra hassle.

NEW AND COMPLEX SECTION

Facebook's React architecture allows developers to create completely self contained elements, as well as think of those elements as state machines. A React Component has 2 different ways to go about variables: props and states. Props are external variables that can be passed in. Where state variables are variables that are self contained in the element. Most uses of react have external functions that can be passed in as props and then executed by the component. Props cannot be changed within the component, whereas state variables can be. State variables must be changed with the setState() method that prevents multiple asynchronous threads from creating write-read errors.

The second part of React that makes it so interesting is that it introduces JSX. JSX is a javascript preprocessor that allows the user to use XML notation that gets converted to Javascript. XML notation is the same as HTML notation allowing the programmer to have to think less. As opposed to having to write a large statement such as this: return React.createElement("div",null,"Hello ",this.props.name); One can simply place: return <div>Hello {this.props.name}</div>; Instead. Another interesting part of React is that CSS Styles are stored as associative arrays as opposed to CSS strings.

Interesting Points Section

1. Notice that the elements of Square and how props are passed into that Component
2. Notice that to change the values of state variables, besides the constructor, one has to use the setState() method. This is an asynchronous call and therefore reading the value of the state variables right after the call will not have the updated state. So, it can be seen that the second parameter to the function is another function that will get called after the setState method completes.
3. Notice the CSS styles are associative arrays and not strings
4. Notice the JSX notation that allows Components to be represented as HTML/XML elements and have variables passed as parameters
5. Check that Tic-Tac-Toe works correctly and all win cases are correct
6. The win message is shown
7. Cats game message is shown

How to install and run - Taken from README.md

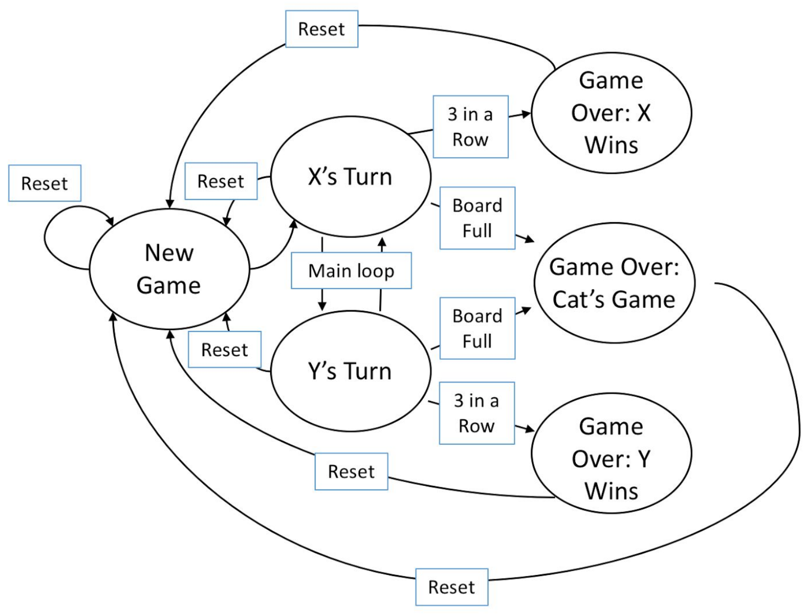
1. Download file and unzip
2. Install homebrew
3. Make sure xcode command line tools is installed
3. `brew install node`
4. Run `npm` to make sure npm is installed
5. If that doesn't work make sure this is in your bash_profile: `export PATH="$HOME/.node/bin:$PATH"`
6. Repeat step 5
7. navigate to project directory
8. run `npm start`
9. Navigate to localhost:3000

BLOOM'S TAXONOMY SECTION

Web Apps as State Machines

React’s UI implementation allows developers to think of their web apps as finite state machines, determining transitions between those states whenever a user interacts with the app. By viewing the app as a series of states, we were able to specify exactly what we wanted each state to look like, by changing which HTML segments were returned upon rendering a state. A simple example of this is the game over screen. Instead of calculating who the winner was, we were able to determine the winner based on the state of the app and a check to see if there were three in a row. Then, we returned the game over screen with the determined winner.

Tic Tac Toe: State Diagram



Props vs States

React has two primary methods for keeping track of data: Props and States. States can be changed with the setState method described above; however, Props are intended to be treated as immutable. These two types of data storage was initially confusing, as it was not immediately clear which type we should use in a given scenario. We learned that since States are changeable, they work well for storing any data that determines the current state of an object (such as keeping track of the player whose turn it is). Props, on the other hand, are useful for passing information from parent elements to child elements, and can even be used to call methods that run in the parent element from the child element.

One example of this functionality was in the Squares on the Tic-Tac-Toe board. When a user clicks a square, we needed the board (the parent element) to update its state and run any checks for a game over, while also updating the Square (the child element) to display the correct value. So, we passed in a method from Main as one of the Square's Props, then called that method whenever a Square was clicked. Doing so allowed us to update the States of the board and the Squares at the same time, without creating any State transition errors.

Example Code:

```
handleButtonPressed(number) {
        if (this.state.gameOver) {
            console.log("Game already won. You cannot move.");
            return;
        }

        var newState = this.state.values;
        if(this.state.playerTurn === 1) {
            newState[number-1] = 1;
            var gameWon = this.checkWin();
            this.setState({values: newState, playerTurn: 2, gameOver: gameWon ||
this.checkCatsGame()});
        }
        else {
            newState[number-1] = 2;
            this.setState({values: newState, playerTurn: 1, gameOver:
this.checkWin() || this.checkCatsGame()});
        }
    }
```

[The method handleButtonPressed() exists at the parent component level, and changes the state of the board.]

```
<div style={style} id="board">
    <Square num="1" value={this.state.values[0]} turn={this.state.playerTurn}
click={this.handleButtonPressed.bind(this)}/>
    <Square num="2" value={this.state.values[1]} turn={this.state.playerTurn}
click={this.handleButtonPressed.bind(this)}/>
    <Square num="3" value={this.state.values[2]} turn={this.state.playerTurn}
click={this.handleButtonPressed.bind(this)}/>
```

```
    <Square num="4" value={this.state.values[3]} turn={this.state.playerTurn}
click={this.handleButtonPressed.bind(this)}/>
    <Square num="5" value={this.state.values[4]} turn={this.state.playerTurn}
click={this.handleButtonPressed.bind(this)}/>
    <Square num="6" value={this.state.values[5]} turn={this.state.playerTurn}
click={this.handleButtonPressed.bind(this)}/>
    <Square num="7" value={this.state.values[6]} turn={this.state.playerTurn}
click={this.handleButtonPressed.bind(this)}/>
    <Square num="8" value={this.state.values[7]} turn={this.state.playerTurn}
click={this.handleButtonPressed.bind(this)}/>
    <Square num="9" value={this.state.values[8]} turn={this.state.playerTurn}
click={this.handleButtonPressed.bind(this)}/>
</div>
```

[The handleButtonPressed method is passed to each child element as a Prop, click, along with other props, value and turn.]

```
handleClick() {
        if(this.state.displayValue !== " ") return;

        if(this.props.turn === 1)
            this.setState({displayValue: "X"});
        else
            this.setState({displayValue: "O"});

        this.props.click(this.props.num); //runs handleButtonPressed method in
main
    }
```

[This method, handleClick(), exists in the child element, Square. When the Square is clicked, it updates its values, then runs the handleButtonPressed method passed to it as this.props.click().]