**React Tic Tac Toe**

**Index JavaScript file Index.js**

Index.js is a JavaScript file that connected to the index.html file via the div that contains the id “root”. Everything that Index.js will be display will be within this div. Index.js starts off by importing several React and Redux imports.

importing react module allows for the creation and use of ES6 classes which serve as components for our react application.

importing react-dom allows for react to automatically preform updates to the DOM, especially when new content needs to render.

importing redux allows for the creation of the store and for combining all potential reducers into one alias.

importing react-redux to set up the provider so that the store can be passed down from index.js to its components

importing redux-thunk allows passing in a function as an action. this allows components and have interaction with the store.

We also import our container which houses our component and the reducer that is responsible for the logic of our tictactoe game.

We take all reducers in our application to be represented by one alias, in this case only one reducer, into a mother of all reducers called reducer. Redux has a function called combineReducers that allows for this procedure to take place.

We then create a location call the store where the state of the DOM will be kept and updated for containers to see. Also this allows for Redux Google Chome Dev Tools to be used.

We then render the components using ReactDOM.render and pass down the store through the provider to give it to the container called TTTContainer which will give access to the store to the component called TTT

**Component file TTT.js**

TTT.js takes in react, react-redux and an actions file as its imports.

The TTT class has render() as its only function and it is responsible for display of the game board.

It returns JSX code that is responsible for displaying:

1)The scoreboard,

2)The result of a clicked tile: X or O, its color, animation if it’s a victory

3)A message that states whos turn is next in the game with this.props.message

4)A play again button that only appears upon victory/draw

The containter then allows any changes that may happen to the state to be applied to state.TicTacToe so that it can be updated in the mother or all reducers called reducer. list of actions as well is kept track of in the store.the TTTCOntainer is then exported to be imported by Index.js

**TTT.Action.js**

This actions file just exports two functions which are responsible for two actions, providing an index to the reducer and a alert named ‘move’ to the reducer and an action with an alert named ‘restart’. These actions correspond to number 2 and number 4 respectively on TTT.js

**TTT.reducer.js**

This function is responsible for a multitude of logic that allows the tictactoe to function as it does.

We create a constant called INITIAL\_STATE to be an object that contains the default of so many characteristics of our game.

tttReducer takes in the state and an action as its parameter. it checks to see if the state of gameOver is false and if it is, we move on to a case and switch that checks our string value of our action.type (either move or restart)

if the case is ‘move’ then we will make a “copy” of the current board without mutating it using state.board.slice() with empty parameters on slice and assign it to temp board. We then immediately use this copy board to get the index of the tile of where the player has clicked on the tictactoe game and apply either X or O (state.currentPlayer) to that element.

We then check to see if this new addition of this element would result into a winning board using

check4Winner(tempBoard) and assigning the boolean result to variable called winner.

Check4Winner then checks the 8 winning possiblies with tictactoe and checks to see if the tmpboard has a victory in either of those 8 possibilities. the variable winner initially is assigned to 0 but if there is a victory, it value is assigned to the first element of the three winning cells, which is either an ‘X’ or an ‘O’. this value for winner as well as the array containing the winning cells, is returned together in an array.

Exiting the function, if the winner[0] is not 0, meaning if the the first element in the returned array, which was winner in the check4Winner function, is not its default value of 0, then update the score depending on whether they are an ‘X’ or an ‘O’. Then indicate that this player has won this game. set game finished to true so that the restart button in TTT.js can activate.

Update the results to the state to the store and maintain purity of the function by using Object.assign()

To test for potential Draws, keep track of cells taken by counting up the number of cells that are not null using a reduce function and ternary operators and assign the overall result to a variable called cellsTaken.

if cellsTaken had all of its cells taken by having a value of 9 in TicTacToe, then create a message sayin that it is a draw and that the game is finished. Then update these results of the state to the store.

To allow the switching from ‘X’ to ‘O’ use a ternary operator to help detect Whether the current player is ‘X’ or ‘O’. If it is either, make it so that it switches to the opposite. then update the store.

at the very end, do default update to the store, even if nothing happens.