ANMOL GUPTA | 005723941 | chorus-lapilli.txt | Assignment 3

BUILDING THE APP
---
1. Setup Local Development Environment
    a. Use 'npx create-react-app chorus-lapilli' to create a new react project
    b. Delete existing files in src folder and create index.css and index.js files
    c. Copy the tic-tac-toe.txt code from the React tutorial into these files and use 'npm start' to run code in the file. Make sure to include the React imports.

    * css file is made already and isn't a focus in tutorial.

> TESTING: We see the tic-tac-toe grid in the browser. (http://localhost:3000/)

2. Limiting Pieces
    a. Let's keep a count of the number of pieces by initializing a variable 'numPieces' in the constructor of the board and incrementing it by 1 in the handleClick function until it equals 6.
    > TESTING: The board only allows 6 pieces (3 Xs and 3 Os).

    b. Functionality for less than 6 pieces is the same as tic-tac-toe so we shift that logic into an if statement for less than 6 pieces and shift the check for no overlap of pieces into the body.
    > TESTING: Works correctly.

3. Dealing With Selection and Move
    a. First, we can get the selection process working by storing the piece and index using a 'storePiece' and 'storeIndex' variable in the body of the if condition. Use a console.log statement to see if this works. Make sure to initialize variables in the constructor.
    > TESTING: When we click a square, the console displays ' "clicked " + squares[i] + " at index:" + i) ' correctly.

    b. Next, to implement selection of a blank square to place a piece in, let's make the else if condition check if the square is null and the player is the current selected piece.
    > TESTING: When we click a square, the console displays ' "clicked " + squares[i] + " at index:" + i) ' correctly.

    c. Now, in order to make a valid move, we can only go horizontally, vertically, and diagonally, so we can make a function in Board 'isAdjacentCell(i, j)' to check if moving from i to j is a valid move by comparing their values for all cases. We can now have and if check calling this function in the condition of clicking a blank square. Write a console log to mark valid move.
    > TESTING: When we click a square with a piece and then a blank square, the console displays 'valid move' for two adjacent squares.

d. Now that we know we are able to make valid moves, let's set the new state to show this change. We can set squares for index i as the piece we are moving and set the old square value as null. Then we can setState to update the squares and switch to the next player.

> TESTING: When we click a square with a piece and then a blank square, if the move is valid, the X disappears from the old position and appears in the newly selected position.

4. Correcting Center Square Functionality

Since the a piece cannot remain on the center square for more than one turn unless it is a winning move, we need to implement this functionality. We can do this by making an if check under the isAdjacentCell condition to check whether the current center square piece is the same as the one from the last turn.

a. If it is we can create tempSquares and update it with the new positions of the pieces if the move is valid. Then we can check: if tempSquares will not be a winning move when the piece to be moved is not the center piece, we must return. This is because this is not a valid move.

b. If this is not true, the move is valid and we can continue updating our board as we did before.

> TESTING: The game now does not allow movement of another piece if the center piece was there for the previous turn unless it is a winning move.

> To do a dry run of the app we do: npm --dry-run. This gives us:

```
anmol@Anmols-MacBook-Air chorus-lapilli % npm --dry-run
npm <command>

Usage:

npm install        install all the dependencies in your project
npm install <foo>  add the <foo> dependency to your project
npm test           run this project's tests
npm run <foo>      run the script named <foo>
npm <command> -h   quick help on <command>
npm -l             display usage info for all commands
npm help <term>    search for help on <term>
npm help npm       more involved overview

All commands:

    access, adduser, audit, bin, bugs, cache, ci, completion,
    config, dedupe, deprecate, diff, dist-tag, docs, doctor,
    edit, exec, explain, explore, find-dupes, fund, get, help,
    hook, init, install, install-ci-test, install-test, link,
    ll, login, logout, ls, org, outdated, owner, pack, ping,
    pkg, prefix, profile, prune, publish, query, rebuild, repo,
    restart, root, run-script, search, set, set-script,
    shrinkwrap, star, stars, start, stop, team, test, token,
    uninstall, unpublish, unstar, update, version, view, whoami

Specify configs in the ini-formatted file:
```

```
    /Users/anmol/.npmrc
or on the command line via: npm <command> --key=value

More configuration info: npm help config
Configuration fields: npm help 7 config

npm@8.19.2 /usr/local/lib/node_modules/npm

> To run the app we use: npm start

>> We've built the game correctly!
```

How to open the app

1. Download the gzipped tarball chorus-lapilli.tgz and change your directory to the package that is downloaded.
2. Run 'npm i' to install some required npm dependencies.
3. Run 'npm start' to run the React app. This opens http://localhost:3000/ on which we can play the game.

How to play the game: Game Rules

1. The game board with 9 squares is initially empty and it is player X's turn first.

2. Player X makes the first move by clicking on any square they desire. Now, it is player O's turn and they make a move similarly. The player whose turn it is will be indicated by the text above the board.

3. Player's alternate in turns until they each have 3 pieces on the board, a total of 6 pieces on the board. In case during these turns, a player achieves a winning combination, like tic-tac-toe, they will win the game. A winning combination is when 3 of the same type of pieces is in a row, column, or diagional. Hence, the game follows the same rules as tic-tac-toe for the first 6 moves.

4. Now, the player will not be able to generate new pieces on the board and will need to move existing pieces to the desired location. To move a piece, click on the piece you wish to move and then click on the empty square to which you want to move it to.

5. We can only move the piece to a valid square which is determined by the following conditions:
   a. You must instead move one of your existing pieces to an empty square that is adjacent vertically, horizontally, or diagonally. Therefore, after your third move you always occupy three squares.
   b. If you have three pieces on the board and one of your pieces is in the center square, you must either win or vacate the center square in your next move.

6. When a player achieves a winning combination, the screen displays text indicating the winning player.
