Julia Vann

Tic-Tac-Toe Write-up

https://tictactoe-julia.herokuapp.com/

Introduction

When first looking at the requirements for this tic-tac-toe app, I came up with a list of things that I considered requirements and another list that I considered my stretch goals. My basic requirements included:

- Having a top bar to state the app title
- Giving the user a choice between playing as X or O
- The 3x3 tic-tac-toe board where a user plays against the computer
- A section showing the wins, losses, and ties for the current browser session
- A popup stating whether the user won, lost, or tied

The list of stretch goals I had included:

- Having a small backend to allow users to login and register in order to save their score data to be accessed at any time.
- Make the app cloud hosted
- Have two difficulty modes

After coming up with my requirements and stretch goals, I had to make decisions on which technologies I wanted to use. ReactJS was what I decided to use for most of the app development as I have previous React experience and it allowed for more simplified frontend development. I also used React-Bootstrap to help with UI design. This is what I used for all the frontend development including the tic-tac-toe logic and app design. For my stretch goals, I decided I would use MongoDB and NodeJS for the backend as well as Heroku for cloud hosting as those were most familiar to me.

Approach

In terms of my approach, I had to first set up my basic app structure using create-react-app, then I started in order of my requirements. To make the top bar, I used Navbar in React-Bootstrap to create a simple top bar component containing the name of the app, “Tic-Tac-Toe”. I also used CSS to make the font bigger. I then moved on to my next component named “choice”. Choice is a component I made to allow users to choose whether they want to play as X or O, with X always going first. To do this I used a simple HTML header to state “Choose X or O” and two buttons for choosing X or O as well as an HTML statement to specify to the user that X always goes first.

After a user clicks on either X or O, the choice component is replaced with the tic-tac-toe board and a small section that shows their wins, losses, and ties for the current browser session. The way the board works is if the user clicks X, they choose where to go first, if they choose O the computer randomly places an X on the board. This goes back and forth with a user selecting the

placement of their X or O and the computer using Math.random() to place its X or O randomly until a win condition is met. The board proved to be a challenge when deciding on how to implement it. In the end, I decided to use React-Bootstrap's ButtonGroup component to create a 3x3 grid of adjacent buttons. Then when a player clicks on one of the 9 buttons, as long as it is not occupied already, the player marks their chosen X or O onto the button. While I liked this implementation idea, I did run into the challenge of re-rendering the ButtonGroup board whenever a player or the computer marks a cell. To overcome this challenge, I used an array of nine objects to represent the state of each board cell. Each cell object contains an integer value representing whether the cell is unmarked, marked by X, or marked by O, and another integer value representing the location of the cell on the board. Then to make the array of objects more usable by the ButtonGroup when deciding what to re-render, I then decided to split the array of objects into three subarrays of three objects each, more accurately representing the 3x3 ButtonGroup grid. This array split was calculated after every turn to ensure that the ButtonGroup grid re-renders correctly.

A win condition is met when three of the same match on the board. My approach to determining win condition is having a 2D array that includes the all the groups of three locations that make up a win condition. I then check after each turn to see if a win condition is met by either the user or the computer. If there is, the winning/losing modal is popped up. This modal states whether the user won or lost depending on whether the user or the computer met the win condition. If the game completes without ever having a win condition, it is considered a tie and the tie modal pops up.

Once a game is completed, the user will then see that either their wins, losses, or ties has increased. This was done using React states and React hooks. The user may then click restart game to go back to the choice screen to play again. Since their current score is saved in React state, it will be saved until they exit or reload the session.

Stretch Goals Approach

After the basics of my app were done, I wanted to move on to my stretch goals. The goal I thought to be most important was cloud hosting. To do this, I used git to deploy my application to Heroku. There were a few challenges I faced here with getting errors when trying to deploy. After some time looking at it, I learned that my issues were stemming from my app's file structure. So, my solution ended up being to change the formatting of my app so that the backend files were in the root directory rather than an isolated folder.

I then moved on to setting up a small backend. The main reason for setting up this backend was so that users could register and login to the app and have their game scores saved even after exiting the session. So, whenever they relogged into the app they could see how many total games they have won, lost, and tied against the computer. To do this, I used MongoDB to set up one database collection called User. The User collection stores a user's username, encrypted password, wins, losses, and ties.

Three APIs were needed to be able to use the database. I used NodeJS and ExpressJS to be able to create login, register, and updateScore APIs. The login and register APIs used bcrypt to

encrypt and hash the passwords that users create and login with. They also each had a couple error checks to ensure unique usernames and correct login information. In order for a user to be able to login or register, I added login and register React-Bootstrap buttons to the top bar that will show the login or register modals when clicked on. Once a user does login or register, those buttons are no longer shown are instead replaced with some text stating that the user is logged in. The updateScore API took a user's id and an integer to signify whether wins, losses, or ties needs to be incremented and it increments it by one each time it is called. This score is displayed on the frontend by calling the login API and using its return values (wins, losses, ties) to overwrite the React states holding the previously defined scores that did not require an account.

My next goal was to come up with a small algorithm to be able to have two difficulty modes. My first difficulty mode is "easy" and relies on the computer choosing a random unused square to add its symbol to; this is what was used in my basic requirements implementation. To start my "hard" difficulty algorithm, I first had to think about my techniques for winning a game of tic-tac-toe. I always try to start in a corner, so if the computer is going first, it always starts in the top left corner. Then the computer tries to go in the corner again the next turn it has. If I go second and my opponent goes in a corner I would choose to go in the middle, so the computer does the same. If the user chooses any spot but the corners, the computer will choose the top left corner. After these first turns, the computer will start checking if there is almost a win by either player. This was done by having an array with all possible locations of two out of three matching symbols as well as an empty third location to complete that match. If the two first locations have the same symbol and the third location is empty, the computer will go there. This results in either blocking a win for the opponent or a win for the computer. If there is no almost win, the computer will place its X or O in a random unused place on the board.

Conclusion

This assessment proved to be a fun challenge to accomplish. I feel I was able to achieve all the goals that I set for myself including the stretch goals. There were some aspects that proved to be a challenge to me such as the board buttons and deployment, but I feel like I learned more about how React and web applications as a whole work through solving those challenges. If there were some things I would choose to work more on for this app, it would be making the UI design even better. An example would be maybe having some sort of animations when a user wins or losses. With UI design, I feel like the work is almost never done and small changes can be made to keep making it look better whether it be through Bootstrap or straight CSS. Another future goal for this app could be refining the hard difficulty algorithm more and even adding a few more difficulties. There are multiple different methods that people have for tic-tac-toe, so having different algorithms could be fun. Overall, I enjoyed the process of coding this assessment.