Homework 2

In this assignment, you will implement a multiplayer tic-tac-toe game that can be played over the Internet through a browser. Tic-tac-toe is a two-player game played on a 3 × 3 board, in which the players take turns to place pieces on the board. The first player to get three-in-a-row wins.

The assignment consists of two parts: the server and the client. We will use Node.js to build a server that will send game state information to the client and keep track of moves sent by the client. We will then use AJAX to build a client that can communicate with the server and display the game interface to the user.

The starter code gives you a working single-player implementation. Your job is to modify it such that two players can play it over the internet.

Getting Started

Download the starter code here.

You will need to install Node.js for your platform. Visit the Node.js website and follow the instructions for your platform.

To run the starter code, open your terminal, go to the root of the project folder and run `npm install`. This should install the required packages for the project. Next, run `node server.js`. You should see a message: "Listening for new connections on http://localhost:4000/". Open your browser and go to http://localhost:4000/. If everything worked fine, you should see your game website.

Server

The logic for the server is contained within the file `server.js`.

The client will send information to and receive information from the server via GET HTTP request. The client will make a GET HTTP request to one of the endpoints (i.e. URLs) below. Any additional information is sent as query parameters. The server will then send back a JSON object that represents its response.

The following endpoints are implmented for you:

`/reset`
: Resets the game. Initially, the board is empty, and the `"x"` player starts first.

`/board`
: Gets the board as a JSON object.

`/turn`
: Gets the name of the current player whose turn it is. The returned value can be either `"x"`, `"o"` or `""` (empty string). If the returned value is an empty string, it means the game

has ended.

You will need to edit `server.js` implement the `/move` endpoint, which is used by clients to send player moves to the server.

This endpoint should be passed three GET `query` parameters: `row`, `col`, and `player`. Upon receiving the move, the server should first check if the move is legal. If the move is illegal, send the JSON reply `false` and do nothing else. If the move is legal, send the JSON reply `true`, save it on board, and update whose turn it is. If the game has ended, set turn to the empty string `""`. Otherwise, set `turn` to the next player.

To test your code, run your server, then open your browser. Navigate to `/board`. You should see an empty 3 × 3 array. Now navigate to `/move?row=0&col=0&player=x`. Navigate back to `/board`. You should see the 3 × 3 board with a cross in the first square. You can make more moves by navigating to `/move` with appropriate parameters, and then check the results at `/board`. If you need to clear the board, visit `/reset`.

You can also use the provided unit test at `/test.html` to test your code. The page will run a series of unit tests. If any tests fail (red) or the tests do not finish, there is a problem with the implementation. If the tests all succeed (green), then you're done.

Hints:

- Use the `gameEnded()` helper function to check if the game has ended.
- HTTP GET query parameters show up as properties of `req.query` (e.g. `req.query.row`, `req.query.col`, `req.query.player`)
- HTTP POST query parameters show up as properties of `req.body` (e.g. `req.body.row`, `req.body.col`, `req.body.player`)
- Use `JSON.stringify` to transform values to JSON strings that can be sent in the response.
- Use `res.send()` and `res.end()` to send the HTTP response to the browser.
- You may wish to refer to the Express documentation.
- The reference solution takes about 20 lines.

Client

The client implementation is found in `/public/game.js` and `/public/game-ui.js`. You will only need to edit `/public/game.js`. You can refer to `/public/game-ui.js` for helpful functions that you can use.

The user interface has already been implemented for you in `/public/game-ui.js`. The primary ones that you will need are:

`new gameUI(elem, player)`
: Initializes a new gameUI object. Takes two parameters. `elem` can be a DOM element, jQuery collection or jQuery selector. The game UI will be initialized in `elem`. `player` should be either `"x"` or `"o"`, and represents whether the user is playing as player X or player O.

`waitForMove()`

Waits for the next legal player move, and then calls the callback function (which is specified by the `callback` property).

`setMessage(message)`
: Takes a string as the only parameter. Displays the string to the user. Use this for displaying game information, such as whose turn it is, or who has won the game.

`setBoard(board)`
: Takes a 3 × 3 array and updates the UI to reflect the current board.

`callback`
: `callback` is a function that will be called by the interface whenever the user makes a move. The callback function should have the signature `callback(row, col, player)`, where `row` and `col` are the coordinates of the chose sequare, and `player` is the current player, which is either `"x"` or `"o"`.

`player`
: `player` is a read-only property that represents whether the user is playing as player X or player O. It will be equal to either `"x"` or `"o"`.

`ended` and `winner`
: `ended` and `winner` are read-only properties. After you call `setBoard`, these properties will be updated to reflect whether the game has ended. `ended` will be true if the game has ended, and false otherwise. `winner` will be equal to `"x"` or `"o"` if player X or player O wins, respectively. It will be equal to `""` (empty string) if the game is a tie.

You should write an initalization function that will be called at the start of the game. The intialization function should get the current board from the server, and display it. It should also get the turn from the server, and take the appropriate next action (prompt user for move, wait for opponent, or notify the user that the game has ended).

You should also write an callback function that will be called whenever the user makes a move. The callback function should send the player's move to the server. It should then take the appropriate next action (wait for opponent, or notify the user that the game has ended).

At each step of the game, you should also display an appropriate message to the user. The required messages are:

- "It is your move."
- "Sending your move…"
- "Waiting for opponent…"
- "The game has ended."

Hints:

- You should write helper functions that provide functionality that you need to use in more than one place. For instance, the logic for waiting for the opponent's move can be placed in helper function.
- To send the player's move to the server, make an AJAX call to `/move` with the appropriate parameters.
- Use `$.getJSON` to send and receive information for the server. If you need more control over your AJAX calls, you can also use `$.get`, `$.post` and `$.ajax`.
- To wait for the opponent to respond, create a timer that makes an AJAX call to `/turn` every second. Stop when `/turn` returns the current player (it is the user's turn) or when

`/turn` returns the empty string (the game is over).

- Use `setInterval` and `clearInterval` to start and stop timers.
- Remember to turn off the timer when it is not needed. You should not be making AJAX calls unnecessarily.
- You may wish to refer to jQuery's AJAX documentation.
- The reference solution takes about 55 lines.

Extra Credit

You can get up to 10% of the assignment (i.e. 4% of the final grade) for implementing extra credit. It should be emphasized that extra credit is *entirely optional*.

- The assignment asks you to use the HTTP GET method for the `/move` to make debugging easier for you. Since `/move` changes state on the server, it should actually be implemented using the HTTP POST method instead. You could change the server and client to use the correct HTTP method.
- The assignment only shows the current game state, but not any of the previous moves. Modify the server to keep track of previous moves and display them in a log.
- The assignment asks you to use HTTP polling. This is undesirable for many reasons: it produces heavy server load from repeated requests, and it has a latency of up to one second. You could implement an alterantive transport method, such as Socket.IO (recommended) or Comet.
- The assignment does not use any database, so the game state will be lost when the server is shut down. To add persistence, save the game state to a database. (This was not covered in class so you should talk me during office hours or do your own reading. I recommend using the mongoose library with MongoDB hosted on MongoHQ).

If you implement extra credit, your implementation is *allowed* to break the automated unit tests. However, to a human player, it must still contain all the basic functionality as a basic implementation.

Submission

To submit, create a compressed .zip archive containing your homework directory. Name the file "cs98si_hw2_username.zip". Email the archive to maiyifan "at" stanford "dot" edu with the subject line "CS98SI HW2 username".

This assignment is due on June 3 (Monday) at 3.15pm. If you use one late day, it will be due on June 5 (Wednesday) at 3.15pm. If you use two late days, it will be due on June 7 (Friday) at 3.15pm.