Stoic Web Developer Programming Challenge

Thank you for taking time to complete this development challenge. It is important for our team to evaluate your ability to write code, understand development tasks, and see how you innovate and solve problems. This task is not meant to be 100% complete, but is used to see what problems you solve and how you may set yourself up for success, if you were to complete all the tasks in the challenge.

Expectations

You will create web code that builds upon the base code provided to complete sets of tasks. Our expectation is that you **spend no more than 4 hours developing** your solutions and **return your solution within 72 hours** to the Stoic team upon receipt of the challenge. You may host your files online, but *we require that your code is zipped and emailed or placed into a git repository and the link sent*.

Should you have any changes to how the code is built or run, please include a brief explanation of the build/run steps that may be required. Additionally, if you wish to briefly outline time spent working on your challenge, it would be helpful for us to understand your process.

Prompt

You are to start building out a chess game in HTML, CSS, and JavaScript. While we provided the base framework, it is up to you to draw the board and pieces and get the initial board layout. We have outlined the required tasks and some future tasks that you should be aware of, and if you have time, optionally begin working on.

Technical Requirements

While the provided base of code is not exact to what Stoic uses, it bears some similarities to how we construct our UI elements. The solution must:

- Use ES6 standards where possible
- Use Custom Elements (Also known as web components) where necessary
- Use Custom Element attributes for state and state change reactions (e.g. the position of a piece on the board)
- Run in Internet Explorer Edge or a Chromium browser (Chrome/Brave/etc)

You may continue to write your code the way we have set it up or optionally create a node project (if it is better suited for you). We are not grading on the best build environment, but on the solutions in the web code.

How to Run the Code

Since we are using JavaScript modules, you will need to run a simple server and access the chess.html page through that server. Some examples of how to run a server are:

- With Node package http-server: `npx http-server ./`
- With python server
  - `python -m SimpleHTTPServer`
  - Python 3 may require: `py -m http.server`
- A simple http server program like Rebex, lighttpd, HFS or similar (use at your own risk)

Challenge Tasks

Required Tasks

Please do your best to complete these tasks in the 4 hour time. This will be what we are looking for when you return the code.

- Draw the basic chess board squares 8x8 (use colors of your preference)
- Draw the basic chess setup of the 16 pieces per side (see reference image below)
  - Use images or Unicode (https://www.i2symbol.com/symbols/chess) for pieces
- Animate the display of the chess pieces (your animation of choice, but make the initial setup look and be engaging)
- Create a button that triggers an event to “reset” the pieces and show the setup animation again

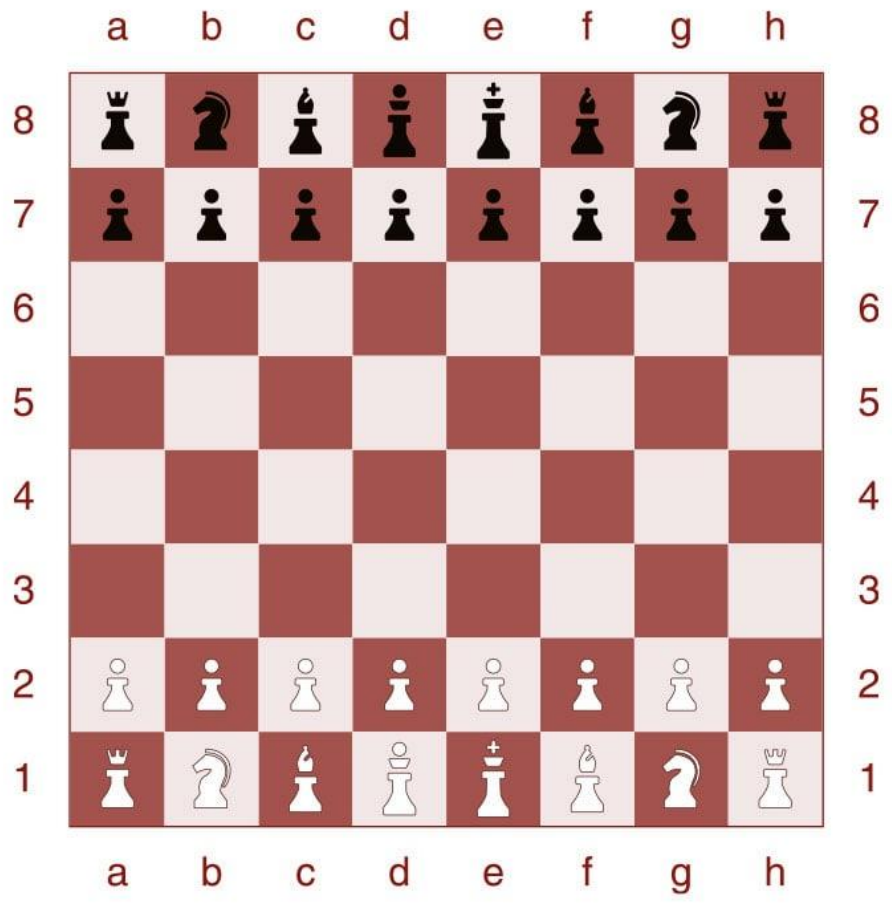
Extra Tasks

Should you have time to spare, feel free to choose some of the tasks below. Please stop at the 4 hour mark, so choose tasks you think may fit!

- Simple piece movement (via move event - move the piece in the “from” location to the “to” location)
  - Animate the move
  - Respond with an error message if there is no piece in the “from” location or if the move is invalid (not a spot on the board)
- Advanced movement
  - Validate the move based on the board state and piece movement rules
  - Employ proper chess notation for moves (e6 or ke6 vs to/from notation)
- Mouse control
  - Allow clicking of a piece, followed by clicking a space on the board to move the piece

    - If you have Advanced movement, consider showing valid move spaces (outline, color, or other effect to show valid locations)
- Add piece taking
    - If the move places the piece on a shared space of the opposing team, remove the opposing piece
- Set the board state from an array
    - Implement a system to translate an array of pieces into the board state (basically, set up an in-progress game)
- Create a move log
    - Log movements and piece exchanges to the screen
- Advanced rules
    - Pawn promotion, castling, and en passant
- Local storage
    - Store board state and move log (if implemented)
    - Restore board state and move log on refresh

Beginning board setup


a b c d e f g h
8 7 6 5 4 3 2 1